

SPRINGER
REFERENCE

Sherif Sakr
Albert Y. Zomaya
Editors

Encyclopedia of Big Data Technologies

 Springer

Encyclopedia of Big Data Technologies

Sherif Sakr • Albert Y. Zomaya
Editors

Encyclopedia of Big Data Technologies

With 429 Figures and 54 Tables



Springer

Editors

Sherif Sakr
Institute of Computer Science
University of Tartu
Tartu, Estonia

Albert Y. Zomaya
School of Information Technologies
Sydney University
Sydney, Australia

ISBN 978-3-319-77524-1 ISBN 978-3-319-77525-8 (eBook)
ISBN 978-3-319-77526-5 (print and electronic bundle)
<https://doi.org/10.1007/978-3-319-77525-8>

Library of Congress Control Number: 2018960889

© Springer Nature Switzerland AG 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

In the field of computer science, data is considered as the main raw material which is produced by abstracting the world into categories, measures, and other representational forms (e.g., characters, numbers, relations, sounds, images, electronic waves) that constitute the building blocks from which information and knowledge are created. In practice, data generation and consumption has become a part of people's daily life especially with the pervasive availability of Internet technology. We are progressively moving toward being a data-driven society where data has become one of the most valuable assets. Big data has commonly been characterized by the defining 3V's properties which refer to huge Volume, consisting of terabytes or petabytes of data; high in Velocity, being created in or near real time; and diversity in Variety of form, being structured and unstructured in nature.

Recently, research communities, enterprises, and government sectors have all realized the enormous potential of big data analytics, and continuous advancements have been emerging in this domain. This Encyclopedia answers the need for solid and comprehensive research source in the domain of Big Data Technologies. The aim of this Encyclopedia is to provide a full picture of various related aspects, topics, and technologies of big data including big data enabling technologies, big data integration, big data storage and indexing, data compression, big data programming models, big SQL systems, big streaming systems, big semantic data processing, graph analytics, big spatial data management, big data analysis, business process analytics, big data processing on modern hardware systems, big data applications, big data security and privacy, and benchmarking of big data systems.

With contributions of many leaders in the field, this Encyclopedia provides the reader with comprehensive reading materials for a large range of audiences. It is a main aim of the Encyclopedia to influence the readers to think further and investigate the areas that are novel to them. The Encyclopedia has been designed to serve as a solid and comprehensive reference not only to expert researchers and software engineers in the field but equally to students and junior researchers as well. The first edition of the Encyclopedia contains more than 250 entries covering a wide range of topics. The Encyclopedia's entries will be updated regularly to follow the continuous advancement in the

domain and have up-to-date coverage available for our readers. It will be available both in print and online versions. We hope that our readers will find this Encyclopedia as a rich and valuable resource and a highly informative reference for Big Data Technologies.

Institute of Computer Science
University of Tartu
Tartu, Estonia

Sherif Sakr

School of Information Technologies
Sydney University
Sydney, Australia
Jan 2019

Albert Y. Zomaya

List of Topics

Big Data Integration

Section Editor: *Maik Thiele*

Data Cleaning
Data Fusion
Data Integration
Data Lake
Data Profiling
Data Wrangling
ETL
Holistic Schema Matching
Integration-Oriented Ontology
Large-Scale Entity Resolution
Large-Scale Schema Matching
Privacy-Preserving Record Linkage
Probabilistic Data Integration
Record Linkage
Schema Mapping
Truth Discovery
Uncertain Schema Matching

Big SQL

Section Editor: *Yuanyuan Tian and Fatma Ozkan*

Big Data Indexing
Caching for SQL-on-Hadoop
Cloud-Based SQL Solutions for Big Data
Columnar Storage Formats
Hive
Hybrid Systems Based on Traditional Database Extensions
Impala
Query Optimization Challenges for SQL-on-Hadoop
SnappyData

Spark SQL

Virtual Distributed File System: Alluxio
Wildfire: HTAP for Big Data

Big Spatial Data Management

Section Editor: *Timos Sellis and Aamir Cheema*

Applications of Big Spatial Data: Health Architectures
Indexing
Linked Geospatial Data
Query Processing – k NN
Query Processing: Computational Geometry
Query Processing: Joins
Spatial Data Integration
Spatial Data Mining
Spatial Graph Big Data
Spatio-social Data
Spatio-textual Data
Spatiotemporal Data: Trajectories
Streaming Big Spatial Data
Using Big Spatial Data for Planning User Mobility
Visualization

Big Semantic Data Processing

Section Editor: *Philippe Cudré-Mauroux and Olaf Hartig*

Automated Reasoning
Big Semantic Data Processing in the Life Sciences Domain
Big Semantic Data Processing in the Materials Design Domain

Data Quality and Data Cleansing of Semantic Data	Python
Distant Supervision from Knowledge Graphs	Scala
Federated RDF Query Processing	SciDB
Framework-Based Scale-Out RDF Systems	R Language: A Powerful Tool for Taming
Knowledge Graph Embeddings	Big Data
Knowledge Graphs in the Libraries and Digital Humanities Domain	Big Data on Modern Hardware Systems
Native Distributed RDF Systems	Section Editor: <i>Bingsheng He</i> <i>and Behrooz Parhami</i>
Ontologies for Big Data	Big Data and Exascale Computing
RDF Dataset Profiling	Computer Architecture for Big Data
RDF Serialization and Archival	Data Longevity and Compatibility
Reasoning at Scale	Data Replication and Encoding
Security and Privacy Aspects of Semantic Data	Emerging Hardware Technologies
Semantic Interlinking	Energy Implications of Big Data
Semantic Search	GPU-Based Hardware Platforms
Semantic Stream Processing	Hardware Reliability Requirements
Visualizing Semantic Data	Hardware-Assisted Compression
Big Data Analysis	Parallel Processing with Big Data
Section Editor: <i>Domenico Talia and</i> <i>Paolo Trunfio</i>	Search and Query Accelerators
Apache Mahout	Storage Hierarchies for Big Data
Apache SystemML	Storage Technologies for Big Data
Big Data Analysis Techniques	Structures for Large Data Sets
Big Data Analysis and IoT	Tabular Computation
Big Data Analysis for Smart City Applications	Big Data Applications
Big Data Analysis for Social Good	Section Editor: <i>Kamran Munir</i> <i>and Antonio Pescape</i>
Big Data Analysis in Bioinformatics	Big Data and Recommendation
Cloud Computing for Big Data Analysis	Big Data Application in Manufacturing
Deep Learning on Big Data	Industry
Energy Efficiency in Big Data Analysis	Big Data Enables Labor Market
Languages for Big Data analysis	Intelligence
Performance Evaluation of Big Data Analysis	Big Data Technologies for DNA
Scalable Architectures for Big Data Analysis	Sequencing
Tools and Libraries for Big Data Analysis	Big Data Warehouses for Smart Industries
Workflow Systems for Big Data Analysis	Big Data for Cybersecurity
Big Data Programming Models	Big Data for Health
Section Editor: <i>Sherif Sakr</i>	Big Data in Automotive Industry
BSP Programming Model	Big Data in Computer Network Monitoring
Clojure	Big Data in Cultural Heritage
Julia	Big Data in Mobile Networks
	Big Data in Network Anomaly Detection
	Big Data in Smart Cities

Big Data in Social Networks
Flood Detection Using Social Media
Big Data Streams

Enabling Big Data Technologies
Section Editor: Rodrigo Neves Calheiros and Marcos Dias de Assuncao

Apache Spark
Big Data Architectures
Big Data Deep Learning Tools
Big Data Visualization Tools
Big Data and Fog Computing
Big Data in the Cloud
Databases as a Service
Distributed File Systems
Graph Processing Frameworks
Hadoop
Mobile Big Data: Foundations, State of the Art, and Future Directions
Network-Level Support for Big Data Computing
NoSQL Database Systems
Orchestration Tools for Big Data
Visualization Techniques

Big Data Transaction Processing
Section Editor: Mohammad Sadoghi

Active Storage
Blockchain Transaction Processing
Conflict-Free Replicated Data Types CRDTs
Coordination Avoidance
Database Consistency Models
Geo-replication Models
Geo-scale Transaction Processing
Hardware-Assisted Transaction Processing
Hardware-Assisted Transaction Processing: NVM
Hybrid OLTP and OLAP
In-Memory Transactions
Transactions in Massively Multiplayer Online Games
Weaker Consistency Models/Eventual Consistency

Distributed Systems for Big Data
Section Editor: Asterios Katsifodimos and Pramod Bhatotia

Achieving Low Latency Transactions for Geo-replicated Storage with Blotter
Advancements in YARN Resource Manager
Approximate Computing for Stream Analytics
Cheap Data Analytics on Cold Storage
Distributed Incremental View Maintenance
HopsFS: Scaling Hierarchical File System
Metadata Using NewSQL Databases
Incremental Approximate Computing
Incremental Sliding Window Analytics
Optimizing Geo-distributed Streaming Analytics
Parallel Join Algorithms in MapReduce
Privacy-Preserving Data Analytics
Robust Data Partitioning
Sliding-Window Aggregation Algorithms
Stream Window Aggregation Semantics and Optimization
StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing
TARDiS: A Branch-and-Merge Approach to Weak Consistency

Big Data Security and Privacy
Section Editor: Junjun Chen and Deepak Puthal

Big Data Stream Security Classification for IoT Applications
Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems
Co-resident Attack in Cloud Computing: An Overview
Data Provenance for Big Data Security and Accountability
Exploring Scope of Computational Intelligence in IoT Security Paradigm
Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption
Network Big Data Security Issues
Privacy Cube
Privacy-Aware Identity Management

Scalable Big Data Privacy with MapReduce	Microbenchmark
Secure Big Data Computing in Cloud: An Overview	SparkBench
Security and Privacy in Big Data Environment	Stream Benchmarks
	System Under Test
	TPC
	TPC-DS
	TPC-H
	TPCx-HS
	Virtualized Big Data Benchmarks
	YCSB
Business Process Analytics	Graph data management and analytics
Section Editor: Marlon Dumas and Matthias Weidlich	Section Editor: Hannes Voigt and George Fletcher
Artifact-Centric Process Mining	Feature Learning from Social Graphs
Automated Process Discovery	Graph Data Integration and Exchange
Business Process Analytics	Graph Data Models
Business Process Deviance Mining	Graph Exploration and Search
Business Process Event Logs and Visualization	Graph Generation and Benchmarks
Business Process Model Matching	Graph Invariants
Business Process Performance Measurement	Graph OLAP
Business Process Querying	Graph Partitioning: Formulations and Applications to Big Data
Conformance Checking	Graph Path Navigation
Data-Driven Process Simulation	Graph Pattern Matching
Decision Discovery in Business Processes	Graph Query Languages
Declarative Process Mining	Graph Query Processing
Decomposed Process Discovery and Conformance Checking	Graph Representations and Storage
Event Log Cleaning for Business Process Analytics	Graph Visualization
Hierarchical Process Discovery	Graph Data Management Systems
Multidimensional Process Analytics	Historical Graph Management
Predictive Business Process Monitoring	Indexing for Graph Query Evaluation
Process Model Repair	Influence Analytics in Graphs
Queue Mining	Link Analytics in Graphs
Streaming Process Discovery and Conformance Checking	Linked Data Management
Trace Clustering	Parallel Graph Processing
	Visual Graph Querying
Big Data Benchmarking	Data Compression
Section Editor: Meikel Poess and Tilmann Rabl	Section Editor: Paolo Ferragina
Analytics Benchmarks	(Web/Social) Graph Compression
Auditing	Compressed Indexes for Repetitive Textual Datasets
Benchmark Harness	Computing the Cost of Compressed Data
CRUD Benchmarks	
Component Benchmark	
End-to-End Benchmark	
Energy Benchmarking	
Graph Benchmarking	
Metrics for Big Data Benchmarks	

Degrees of Separation and Diameter
in Large Graphs
Delta Compression Techniques
Dimension Reduction
Genomic Data Compression
Grammar-Based Compression
Inverted Index Compression
RDF Compression
Similarity Sketching

Big Stream Processing

Section Editor: *Alessandro Margara*
and Tilmann Rabl

Adaptive Windowing
Apache Apex
Apache Flink
Apache Kafka
Apache Samza

Continuous Queries
Definition of Data Streams
Elasticity
Introduction to Stream Processing Algorithms
Management of Time
Online Machine Learning Algorithms over
Data Streams
Online Machine Learning in Big Data Streams:
Overview
Pattern Recognition
Recommender Systems over Data Streams
Reinforcement Learning, Unsupervised Methods,
and Concept Drift in Stream Learning
Rendezvous Architectures
Stream Processing Languages and Abstractions
Stream Query Optimization
Streaming Microservices
Types of Stream Processing Algorithms
Uncertainty in Streams

About the Editors



Professor Sherif Sakr is the Head of Data Systems Group at the Institute of Computer Science, University of Tartu. He received his PhD degree in Computer and Information Science from Konstanz University, Germany, in 2007. He received his BSc and MSc degrees in Computer Science from the Information Systems Department at the Faculty of Computers and Information in Cairo University, Egypt, in 2000 and 2003, respectively. During his career, Prof. Sakr held appointments in several international and reputable organizations including the University of New South Wales, Macquarie University, Data61/CSIRO, Microsoft Research, Nokia Bell Labs, and King Saud bin Abdulaziz University for Health Sciences.

Prof. Sakr's research interest is data and information management in general, particularly in big data processing systems, big data analytics, data science, and big data management in cloud computing platforms. Professor Sakr has published more than 100 refereed research publications in international journals and conferences such as the *Proceedings of the VLDB Endowment* (VLDB), *IEEE Transactions on Parallel and Distributed Systems* (IEEE TPDS), *IEEE Transactions on Service Computing* (IEEE TSC), *IEEE Transactions on Big Data* (IEEE TBD), *ACM Computing Surveys* (ACM CSUR), *Journal of Computer and System Sciences* (JCSS), *Information Systems Journal*, *Cluster Computing*, *Journal of Grid Computing*, *IEEE Communications Surveys and Tutorials* (IEEE COMST), *IEEE Software*, *Scientometrics*, *VLDB*, *SIGMOD*, *ICDE*, *EDBT*, *WWW*, *CIKM*, *ISWC*, *BPM*, *ER*, *ICWS*, *ICSOC*, *IEEE SCC*, *IEEE Cloud*, *TPCTC*, *DASFAA*, *ICPE*, and *JCDL*. Professor Sakr Co-authored 5 books and Co-Edited 3 other books in the areas of data and information management and processing. Sherif is an associate editor of the cluster computing journal and Transactions on Large-Scale Data and Knowledge-Centered Systems (TLDKS). He is also an editorial board member of many reputable international journals. Prof. Sakr is an ACM Senior Member and an IEEE Senior Member. In 2017, he has been

appointed to serve as an ACM Distinguished Speaker and as an IEEE Distinguished Speaker. For more information, please visit his personal web page (<http://kodu.ut.ee/~sakr/>) and his research group page (<http://bigdata.cs.ut.ee/>)



Albert Y. Zomaya is currently the *Chair Professor of High Performance Computing & Networking* in the School of Information Technologies, University of Sydney. He is also the *Director of the Centre for Distributed and High Performance Computing* which was established in late 2009. Dr. Zomaya was an *Australian Research Council Professorial Fellow* during 2010–2014 and held the *CISCO Systems Chair Professor of Internetworking* during the period 2002–2007 and also was Head of school for 2006–2007 in the same school.

Prior to his current appointment, he was a Full Professor in the Electrical and Electronic Engineering Department at the University of Western Australia, where he also led the Parallel Computing Research Laboratory during the period 1990–2002. He served as Associate, Deputy, and Acting Head in the same department and held numerous visiting positions and has extensive industry involvement.

Dr. Zomaya published more than 600 scientific papers and articles and is author, co-author, or editor of more than 20 books. He served as the Editor in Chief of the *IEEE Transactions on Computers* (2011–2014). Currently, he serves as a Founding Editor in Chief for the *IEEE Transactions on Sustainable Computing*, a Co-founding Editor in Chief of the *IET Cyber-Physical Systems: Theory and Applications*, and an Associate Editor in Chief (special issues) of the *Journal of Parallel and Distributed Computing*.

Dr. Zomaya is an Associate Editor for several leading journals, such as the *ACM Transactions on Internet Technology*, *ACM Computing Surveys*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Computational Social Systems*, and *IEEE Transactions on Big Data*. He is also the Founding Editor of several book series, such as the *Wiley Book Series on Parallel and Distributed Computing*, the *Springer Scalable Computing and Communications*, and the *IET Book Series on Big Data*.

Dr. Zomaya was the Chair the *IEEE Technical Committee on Parallel Processing* (1999–2003) and currently serves on its executive committee. He is the *Vice-Chair* of the *IEEE Task Force on Computational Intelligence for Cloud Computing* and serves on the advisory board of the *IEEE Technical Committee on Scalable Computing* and the steering committee of the *IEEE Technical Area in Green Computing*.

Dr. Zomaya has delivered more than 180 keynote addresses, invited seminars, and media briefings and has been actively involved, in a variety of capacities, in the organization of more than 700 conferences.

Dr. Zomaya is a Fellow of the IEEE, the American Association for the Advancement of Science, and the Institution of Engineering and Technology (UK). He is a Chartered Engineer and an IEEE Computer Society's Golden Core Member. He received the *1997 Edgeworth David Medal* from the Royal Society of New South Wales for outstanding contributions to Australian science. Dr. Zomaya is the recipient of the *IEEE Technical Committee on Parallel Processing Outstanding Service Award* (2011), the *IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing* (2011), the *IEEE Computer Society Technical Achievement Award* (2014), and the *ACM MSWIM Reginald A. Fessenden Award* (2017). His research interests span several areas in parallel and distributed computing and complex systems. More information can be found at <http://www.it.usyd.edu.au/~azom0780/>.

About the Section Editors



Pramod Bhatotia The University of Edinburgh and Alan Turing Institute, Edinburgh, UK



Rodrigo N. Calheiros School of Computing, Engineering and Mathematics, Western Sydney University, Penrith, NSW, Australia



Aamir Cheema Monash University, Clayton, VIC, Australia



Jinjun Chen School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC, Australia



Philippe Cudré-Mauroux eXascale Infolab, University of Fribourg, Fribourg, Switzerland



Marcos Dias de Assuncao Inria Avalon, LIP Laboratory, ENS Lyon, University of Lyon, Lyon, France



Marlon Dumas Institute of Computer Science, University of Tartu, Tartu, Estonia



Paolo Ferragina Department of Computer Science, University of Pisa, Pisa, Italy



George Fletcher Technische Universiteit Eindhoven, Eindhoven, Netherlands



Olaf Hartig Linköping University, Linköping, Sweden



Bingsheng He National University of Singapore, Singapore, Singapore



Asterios Katsifodimos TU Delft, Delft, Netherlands



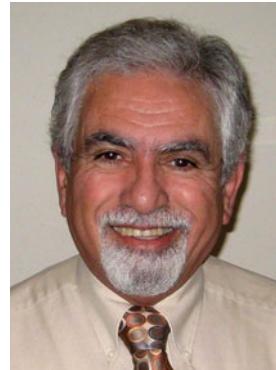
Alessandro Margara Politecnico di Milano, Milano, Italy



Kamran Munir Computer Science and Creative Technologies, University of the West of England, Bristol, UK



Fatma Özcan IBM Research – Almaden, San Jose, CA, USA



Behrooz Parhami Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA, USA



Antonio Pescapè Department of Electrical Engineering and Information
Technology, University of Napoli Federico II, Napoli, Italy



Meikel Poess Server Technologies, Oracle Corporation, Redwood Shores,
CA, USA



Deepak Puthal Faculty of Engineering and Information Technologies,
School of Electrical and Data Engineering, University of Technology Sydney,
Ultimo, NSW, Australia



Tilmann Rabl Database Systems and Information Management Group,
Technische Universität Berlin, Berlin, Germany



Mohammad Sadoghi University of California, Davis, CA, USA



Timos Sellis Swinburne University of Technology, Data Science Research Institute, Hawthorn, VIC, Australia



Domenico Talia University of Calabria, Rende, Italy



Maik Thiele Database Systems Group, Technische Universität Dresden, Dresden, Saxony, Germany



Yuanyuan Tian IBM Research – Almaden, San Jose, CA, USA



Paolo Trunfio University of Calabria, DIMEs, Rende, Italy



Hannes Voigt Dresden Database Systems Group, Technische Universität Dresden, Dresden, Germany



Matthias Weidlich Humboldt-Universität zu Berlin, Department of Computer Science, Berlin, Germany

List of Contributors

Ziawasch Abedjan Technische Universität Berlin, Berlin, Germany

Alberto Abelló Polytechnic University of Catalonia, Barcelona, Spain

Umut A. Acar CMU, Pittsburgh, PA, USA

Biswaranjan Acharya Kalinga Institute of Industrial Technology (KIIT), Deemed to be University, Bhubaneswar, India

Maribel Acosta Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany

Khandakar Ahmed Institute for Sustainable Industries and Liveable Cities, VU Research, Victoria University, Melbourne, Australia

Marco Aldinucci Computer Science Department, University of Turin, Turin, Italy

Alexander Alexandrov Database Systems and Information Management Group (DIMA), Technische Universität Berlin, Berlin, Germany

Khaled Ammar Thomson Reuters Labs, Thomson Reuters, Waterloo, ON, Canada

Carina Andrade Department of Information Systems, ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

Renzo Angles Universidad de Talca, Talca, Chile

Raja Appuswamy Data Science Department, EURECOM, Biot, France

Walid G. Aref Purdue University, West Lafayette, IN, USA

Marcelo Arenas Pontificia Universidad Católica de Chile, Santiago, Chile

Rickard Armiento Linköping University, Swedish e-Science Research Centre, Linköping, Sweden

Julien Audiffren Exascale Infolab, University of Fribourg, Fribourg, Switzerland

Booma Sowkarthiga Balasubramani University of Illinois at Chicago, Chicago, IL, USA

S. Balkir Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

Soumya Banerjee Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, India

Carlos Baquero HASLab/INESC TEC and Universidade do Minho, Braga, Portugal

Ronald Barber IBM Research – Almaden, San Jose, CA, USA

Pablo Barceló Universidad de Chile, Santiago, Chile

Przemysław Błaśkiewicz Faculty of Fundamental Problems of Technology, Wrocław University of Science and Technology; National Science Center, Wrocław, Poland

Guillaume Baudart IBM Research, Yorktown Heights, NY, USA

Sibghat Ullah Bazai Institute of Natural and Mathematical Sciences, Massey University, Auckland, New Zealand

Martin Beck TU Dresden, Dresden, Germany

Alex Behm Databricks, San Francisco, CA, USA

Loris Belcastro DIMES, University of Calabria, Rende, Italy

András A. Benczúr Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary

Konstantina Bereta Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece

Antonio Berlanga Applied Artificial Intelligence Group, Universidad Carlos III de Madrid, Colmenarejo, Spain

Computer Science and Engineering, Universidad Carlos III de Madrid, Colmenarejo, Spain

Laure Berti-Équille Aix-Marseille University, CNRS, LIS, Marseille, France

Milind A. Bhandarkar Ampool, Inc., Santa Clara, CA, USA

Pramod Bhatotia The University of Edinburgh and Alan Turing Institute, Edinburgh, UK

Sourav S. Bhowmick Nanyang Technological University, Singapore, Singapore

Nikos Bikakis ATHENA Research Center, Athens, Greece

Spyros Blanas Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Eva Blomqvist Linköping University, Linköping, Sweden

Matthias Boehm IBM Research – Almaden, San Jose, CA, USA

Paolo Boldi Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Angela Bonifati Lyon 1 University, Villeurbanne, France

Samia Bouzefrane Conservatoire National des Arts et Metiers, Paris, France

Andrey Brito UFCG, Campina Grande, Brazil

Andrea Burattin DTU Compute, Software Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

Rodrigo N. Calheiros School of Computing, Engineering and Mathematics, Western Sydney University, Penrith, NSW, Australia

Mario Cannataro Data Analytics Research Center, University “Magna Græcia” of Catanzaro, Catanzaro, Italy

Paris Carbone KTH Royal Institute of Technology, Stockholm, Sweden

Josep Carmona Universitat Politècnica de Catalunya, Barcelona, Spain

Abel Armas Cervantes University of Melbourne, Melbourne, VIC, Australia

Eugenio Cesario Institute of High Performance Computing and Networks of the National Research Council of Italy (ICAR-CNR), Rende, Italy

Abhishek Chandra University of Minnesota, Minneapolis, MN, USA

Chii Chang Mobile and Cloud Lab, Institute of Computer Science, University of Tartu, Tartu, Estonia

Jinjun Chen School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC, Australia

Ruichuan Chen Nokia Bell Labs, Stuttgart, Germany

Nokia Bell Labs, NJ, USA

Xuntao Cheng Nanyang Technological University, Jurong West, Singapore

Shekha Chenthara Institute for Sustainable Industries and Liveable Cities, VU Research, Victoria University, Melbourne, Australia

Byron Choi Hong Kong Baptist University, Hong Kong, China

Xu Chu School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Carmela Comito CNR-ICAR, Rende, Italy

Raffaele Conforti University of Melbourne, Melbourne, VIC, Australia

Gao Cong School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

Carlos Costa CCG – Centro de Computação Gráfica and ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

Pierluigi Crescenzi Dipartimento di Matematica e Informatica, Università di Firenze, Florence, Italy

Alain Crolotte Teradata Corporation, El Segundo, CA, USA

Natacha Crooks The University of Texas at Austin, Austin, TX, USA

Isabel F. Cruz University of Illinois at Chicago, Chicago, IL, USA

Philippe Cudre-Mauroux eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Renato Luiz de Freitas Cunha IBM Research, São Paulo, Brazil

Emily May Curtin IBM, New York, USA

Alessandro D'Alconzo Austrian Institute of Technology, Vienna, Austria

Guilherme da Cunha Rodrigues Federal Institute of Education Science and Technology Sul-Rio Grandense (IFSUL), Charqueadas, Brazil

Alexandre da Silva Veith Inria Avalon, LIP Laboratory, ENS Lyon, University of Lyon, Lyon, France

Massimiliano de Leoni Eindhoven University of Technology, Eindhoven, The Netherlands

Anna Delin Royal Institute of Technology, Swedish e-Science Research Centre, Stockholm, Sweden

Adela del-Río-Ortega University of Seville, Sevilla, Spain

Gianluca Demartini The University of Queensland, St. Lucia, QLD, Australia

Elena Demidova L3S Research Center, Leibniz Universität Hannover, Hanover, Germany

Benoît Depaire Research Group Business Informatics, Hasselt University, Diepenbeek, Belgium

Rogério Abreu de Paula IBM Research, Rua Tutóia 1157, São Paulo, Brazil

Amol Deshpande Computer Science Department, University of Maryland, College Park, MD, USA

Renata Ghislotti Duarte de Souza Granha Bosch, Chicago, USA

Helena F. Deus Elsevier Labs, Cambridge, MA, USA

Ricardo J. Dias SUSE Linux GmbH and NOVA LINCS, Lisbon, Portugal

Marcos Dias de Assuncao Inria Avalon, LIP Laboratory, ENS Lyon, University of Lyon, Lyon, France

Arturo Diaz-Perez Cinvestav Tamaulipas, Victoria, Mexico

Stefan Dietze L3S Research Center, Leibniz Universität Hannover, Hanover, Germany

Chiara Di Francescomarino Fondazione Bruno Kessler – FBK, Trento, Italy

Chris Douglas Microsoft, Washington, DC, USA

Jim Dowling KTH – Royal Institute of Technology, Stockholm, Sweden

Idilio Drago Politecnico di Torino, Turin, Italy

Maurizio Drocco Computer Science Department, University of Turin, Turin, Italy

Jianfeng Du Guangdong University of Foreign Studies, Guangdong, China

Dirk Duellmann Information Technology Department, CERN, Geneva, Switzerland

Marlon Dumas Institute of Computer Science, University of Tartu, Tartu, Estonia

Ted Dunning MapR Technologies and Apache Software Foundation, Santa Clara, CA, USA

Ahmed Eldawy Department of Computer Science and Engineering, University of California, Riverside, CA, USA

Mohamed Y. Eltabakh Worcester Polytechnic Institute, Worcester, MA, USA

Roberto R. Expósito Computer Architecture Group, Universidade da Coruña, A Coruña, Spain

Dirk Fahland Eindhoven University of Technology, Eindhoven, The Netherlands

Javier D. Fernández Complexity Science Hub Vienna, Vienna University of Economics and Business, Vienna, Austria

Christof Fetzer TU Dresden, Dresden, Germany

Pierdomenico Fiadino Data Science and Big Data Analytics Unit, EURECAT (Technology Centre of Catalonia), Barcelona, Spain

Clark Fitzgerald Department of Statistics, University of California, Davis, CA, USA

George Fletcher Technische Universiteit Eindhoven, Eindhoven, Netherlands

Avrilia Floratou Microsoft, Sunnyvale, CA, USA

Francesco Folino National Research Council, Institute for High Performance Computing and Networking (ICAR-CNR), Rende (CS), Italy

Gianluigi Folino ICAR-CNR (Institute for High Performance Computing and Networks, National Research Council), Rende, Italy

Ellen Friedman MapR Technologies and Apache Software Foundation, Santa Clara, CA, USA

Steffen Friedrich Department of Informatics, University of Hamburg, Hamburg, Germany

Travis Gagie EIT, Diego Portales University, Santiago, Chile

Avigdor Gal Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel

Jesús García Applied Artificial Intelligence Group, Universidad Carlos III de Madrid, Colmenarejo, Spain

Computer Science and Engineering, Universidad Carlos III de Madrid, Colmenarejo, Spain

Alberto Garcia-Robledo Universidad Politecnica de Victoria, Victoria, Mexico

Saurabh Garg School of Engineering and ICT, University of Tasmania, Hobart, TAS, Australia

Alan F. Gates Hortonworks, Santa Clara, CA, USA

Ricard Gavaldà Universitat Politècnica de Catalunya, Barcelona, Spain

Buğra Gedik Department of Computer Engineering, Bilkent University, Ankara, Turkey

Johannes Gehrke Microsoft, Seattle, USA

Jana Giceva Department of Computing, Imperial College London, London, UK

Robert M. Goerge Chapin Hall at the University of Chicago, Chicago, IL, USA

Lukasz Golab University of Waterloo, Waterloo, ON, Canada

Jose-Luis Gonzalez-Compean Cinvestav Tamaulipas, Victoria, Mexico

Anja Gruenheid Google Inc., Madison, WI, USA

Massimo Guarascio ICAR-CNR (Institute for High Performance Computing and Networks, National Research Council), Rende, Italy

Vincenzo Gulisano Chalmers University of Technology, Gothenburg, Sweden

Mehmet Gültas Department of Breeding Informatics, Georg-August University, Göttingen, Germany

Ananth Gundabattula Commonwealth Bank of Australia, Sydney, NSW, Australia

Fuchun Guo Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia

Jayant Gupta Department of Computer Science, University of Minnesota, Minneapolis, MN, USA

- Suyash Gupta** Department of Computer Science, University of California, Davis, CA, USA
- Claudio Gutiérrez** Department of Computer Science and Millenium Institute on Foundations of Data, Universidad de Chile, Santiago, Chile
- Amnir Hadachi** Institute of Computer Science, University of Tartu, Tartu, Estonia
- Maryam A. Haeri** Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
- Martin Hahmann** Databases Systems Group, Technische Universität Dresden, Dresden, Germany
- Rihan Hai** RWTH Aachen University, Aachen, Germany
- Margaret Hamilton** School of Science (Computer Science and IT), RMIT University, Melbourne, VIC, Australia
- Jinguang Han** Department of Computer Science, Surrey Centre for Cyber Security, University of Surrey, Guildford, UK
- Muhammad Hanif** School of Computer Science, National University of Computer and Emerging Sciences, Karachi, Pakistan
- Sergey Hardock** Databases and Distributed Systems Group, TU Darmstadt, Darmstadt, Germany
- Seif Haridi** KTH – Royal Institute of Technology, Stockholm, Sweden
- Olaf Hartig** Linköping University, Linköping, Sweden
- Bernhard Hashhofer** Austrian Institute of Technology, Vienna, Austria
- Manfred Hauswirth** Open Distributed Systems, Technical University of Berlin, Berlin, Germany
- Fraunhofer FOKUS, Berlin, Germany
- Bingsheng He** National University of Singapore, Singapore, Singapore
- Jeffrey Heer** University of Washington, Seattle, WA, USA
- Benjamin Heintz** University of Minnesota, Minneapolis, MN, USA
- Joseph M. Hellerstein** University of California, Berkeley, Berkeley, CA, USA
- Antonio Hernández-Illera** Department of Computer Science, Universidad de Valladolid, Valladolid, Spain
- Jan Hidders** Vrije Universiteit Brussel, Brussels, Belgium
- Shadi M. S. Hilles** Al-Madinah International University, Shah Alam, Selangor, Malaysia
- Volker Hilt** Nokia Bell Labs, Stuttgart, Germany
- Martin Hirzel** IBM Research AI, Yorktown Heights, NY, USA

Erik G. Hoel Environmental Systems Research Institute, Redlands, CA, USA

Nur Farhana Hordri Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia

UTM Big Data Centre, Universiti Teknologi Malaysia, Johor, Malaysia

Katja Hose Aalborg University, Aalborg, Denmark

Jhalak Hota Trident Academy of Technology, Bhubaneswar, India

Yifan Hu Yahoo Research, Oath Inc., New York, NY, USA

Yuchong Hu School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

Fabian Hueske Data Artisans GmbH, Berlin, Germany

Ioana Ileana Paris Descartes University, Paris, France

Antoine Isaac Europeana Foundation, The Hague, The Netherlands

Mahmoud Ismail KTH – Royal Institute of Technology, Stockholm, Sweden

Todor Ivanov Frankfurt Big Data Lab, Goethe University Frankfurt, Frankfurt, Germany

Suresh Jagannathan Purdue University, West Lafayette, IN, USA

Julian Jang-Jaccard Institute of Natural and Mathematical Sciences, Massey University, Auckland, New Zealand

Christian S. Jensen Department of Computer Science, Aalborg University, Aalborg, Denmark

Calvin Jia Alluxio Inc., San Mateo, CA, USA

Peng Jiang Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong

Peiquan Jin School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Alekh Jindal Microsoft, Redmond, WA, USA

Flavio P. Junqueira Dell EMC, Hopkinton, MA, USA

Sean Kandel Trifacta. Inc, San Francisco, CA, USA

Konstantinos Karanasos Microsoft, Washington, DC, USA

Dimitrios Karapiperis School of Science and Technology, Hellenic Open University, Patras, Greece

Jeyhun Karimov DFKI GmbH, Berlin, Germany

Asterios Katsifodimos TU Delft, Delft, Netherlands

- Shadi Khalifa** Queen's University, Kingston, Canada
- Arijit Khan** Nanyang Technological University, Singapore, Singapore
- Zaheer Khan** University of the West of England, Bristol, UK
- Zuhair Khayyat** Lucidya LLC, Jeddah, Saudi Arabia
- Udayan Khurana** IBM Research AI, TJ Watson Research Center, New York, NY, USA
- Sabrina Kirrane** Vienna University of Economics and Business, Vienna, Austria
- Martin Kleppmann** University of Cambridge, Cambridge, UK
- Ryan K. L. Ko** Cyber Security Lab – Department of Computer Science, University of Waikato, Hamilton, New Zealand
- Andreas Koch** Embedded Systems and Applications Group, TU Darmstadt, Darmstadt, Germany
- Levente Kocsis** Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
- Marcel Kornacker** Blink Computing, San Francisco, CA, USA
- Manolis Koubarakis** Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece
- Dhanya R. Krishnan** TU Dresden, Dresden, Germany
- Mirosław Kutyłowski** Faculty of Fundamental Problems of Technology, Wrocław University of Science and Technology; National Science Center, Wrocław, Poland
- Laks V. S. Lakshmanan** The University of British Columbia, Vancouver, BC, Canada
- Patrick Lambrix** Linköping University, Swedish e-Science Research Centre, Linköping, Sweden
- Klaus-Dieter Lange** Hewlett Packard Enterprise, Houston, TX, USA
- Kasper Green Larsen** Aarhus University, Aarhus, Denmark
- Chiew Tong Lau** Nanyang Technological University, Singapore, Singapore
- Duc C. Le** Dalhousie University, Halifax, NS, Canada
- Sander J. J. Leemans** Queensland University of Technology, Brisbane, Australia
- João Leitão** DI/FCT/Universidade NOVA de Lisboa and NOVA LINCS, Lisbon, Portugal
- Henrik Leopold** Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Danh Le-Phuoc Open Distributed Systems, Technical University of Berlin, Berlin, Germany

Haoyuan Li Alluxio Inc, San Mateo, CA, USA

Huanyu Li Linköping University, Swedish e-Science Research Centre, Linköping, Sweden

Lei Li School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD, Australia

Wenhai Li Advanced Computing Research Centre, Data to Decisions Co-operative Research Centre, University of South Australia, Adelaide, Australia

Xiao Li Databricks Inc, San Francisco, CA, USA

Cheng Lian Databricks Inc, San Francisco, CA, USA

Leonid Libkin School of Informatics, University of Edinburgh, Edinburgh, UK

Cheng Liu National University of Singapore, Singapore, Singapore

Hang Liu University of Massachusetts Lowell, Lowell, MA, USA

Huan Liu Data Mining and Machine Learning Lab, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Yi Lu MIT, Cambridge, MA, USA

Michael Luggen eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Berne University of Applied Sciences, Bern, Switzerland

Tariq Magdon-Ismail VMware, Inc, Palo Alto, CA, USA

Amr Magdy University of California, Riverside, California, USA

Fabrizio Maria Maggi University of Tartu, Tartu, Estonia

Adnan Mahmood Telecommunication Software and Systems Group, Waterford Institute of Technology, Waterford, Ireland

Universiti Malaysia Sarawak, Kota Samarahan, Sarawak, Malaysia

Al-Madinah International University, Shah Alam, Selangor, Malaysia

Ahmed R. Mahmood Purdue University, West Lafayette, IN, USA

Nikos Mamoulis Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

Sebastian Maneth Department of Mathematics and Informatics, Universität Bremen, Bremen, Germany

Felix Mannhardt Department of Economics and Technology Management, SINTEF Technology and Society, Trondheim, Norway

Alessandro Margara Politecnico di Milano, Milano, Italy

Andrea Marino Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Fabrizio Marozzo DIMES, University of Calabria, Rende, Italy

André Martin TU Dresden, Dresden, Germany

Niels Martin Research group Business Informatics, Hasselt University, Diepenbeek, Belgium

Miguel A. Martínez-Prieto Department of Computer Science, Universidad de Valladolid, Valladolid, Spain

Norman Matloff Department of Computer Science, University of California, Davis, CA, USA

Hanene Maupas IDEMIA Colombes, Colombes, France

Marco Mellia Politecnico di Torino, Turin, Italy

Jan Mendling WU Vienna, Vienna, Austria

Fabio Mercorio Department of Statistics and Quantitative Methods – CRISP Research Centre, University of Milan Bicocca, Milan, Italy

Mario Mezzananza Department of Statistics and Quantitative Methods – CRISP Research Centre, University of Milan Bicocca, Milan, Italy

Nicolas Michael Oracle, San Francisco, CA, USA

Mart Min Thomas Johann Seebeck Institute of Electronics, Tallinn University of Technology, Tallinn, Estonia

Claudia Misale Cognitive and Cloud, Data-Centric Solutions, IBM TJ Watson Research Center, New York, NY, USA

Sambit Kumar Mishra National Institute of Technology Rourkela, Rourkela, India

Amit Mitra Bristol Business School, University of the West of England, Bristol, UK

Shu Mo Databricks Inc, San Francisco, CA, USA

Nicolas Modrzyk Karabiner Software, Tokyo, Japan

Alistair Moffat School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia

Nurulhuda Firdaus Mohd Azmi Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia

José Manuel Molina Applied Artificial Intelligence Group, Universidad Carlos III de Madrid, Colmenarejo, Spain

Computer Science and Engineering, Universidad Carlos III de Madrid, Colmenarejo, Spain

Henrique Moniz Google and NOVA LINCS, New York, USA

Vincenzo Moscato Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, University of Naples, Naples, Italy

Davide Mottin Hasso Plattner Institute, Potsdam, Germany

Barzan Mozafari University of Michigan, Ann Arbor, MI, USA

Awais Akram Mughal Beijing Institute of Technology, Beijing, China

Kamran Munir Computer Science and Creative Technologies, University of the West of England, Bristol, UK

Jorge Munoz-Gama Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Andrew Musselman Apache Software Foundation, Seattle, WA, USA

Sergi Nadal Polytechnic University of Catalonia, Barcelona, Spain

Sandeep Nagar GD Goenka University, Gurgaon, Haryana, India

Mahsa Najafzadeh Purdue University, West Lafayette, IN, USA

Raghu Nambiar Advanced Micro Devices (AMD), Suwanee, GA, USA

Gonzalo Navarro Department of Computer Science, University of Chile, Santiago, Chile

Faisal Nawab University of California, Santa Cruz, CA, USA

Salman Niazi KTH – Royal Institute of Technology, Stockholm, Sweden

Charalampos Nikolaou Department of Computer Science, University of Oxford, Oxford, UK

Milos Nikolic University of Oxford, Oxford, UK

Zhaojie Niu National University of Singapore, Singapore, Singapore

Martin Nöllenburg Institute of Logic and Computation, TU Wien, Vienna, Austria

Ibrahim Numanagić Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Julian Oppermann Embedded Systems and Applications Group, TU Darmstadt, Darmstadt, Germany

Rasmus Pagh Computer Science Department, IT University of Copenhagen, Copenhagen S, Denmark

Róbert Pálovics Department of Computer Science, Stanford University, Stanford, CA, USA

Jeff Z. Pan University of Aberdeen, Scotland, UK

Gene Pang Alluxio Inc, San Mateo, CA, USA

Alessandro V. Papadopoulos Mälardalen University, Västerås, Sweden

Marina Papatriantafilou Chalmers University of Technology, Gothenburg, Sweden

Paolo Papotti Campus SophiaTech – Data Science Department, EURECOM, Biot, France

Marcus Paradies SAP SE, Walldorf, Germany

SAP SE, Berlin, Germany

Behrooz Parhami Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Miguel Angel Patricio Applied Artificial Intelligence Group, Universidad Carlos III de Madrid, Colmenarejo, Spain

Computer Science and Engineering, Universidad Carlos III de Madrid, Colmenarejo, Spain

Johns Paul Nanyang Technological University, Singapore, Singapore

Valerio Persico University of Napoli “Federico II”, Naples, Italy

Jan Peters-Anders Austrian Institute of Technology, Vienna, Austria

Matthias Petri School of Computing and Information Systems, The University of Melboure, Melbourne, VIC, Australia

Ilia Petrov Data Management Lab, Reutlingen University, Reutlingen, Germany

Eric Peukert University of Leipzig, Leipzig, Germany

Thye Way Phua Cyber Security Lab – Department of Computer Science, University of Waikato, Hamilton, New Zealand

Giulio Ermanno Pibiri Department of Computer Science, University of Pisa, Pisa, Italy

Antonio Picariello Università di Napoli Federico II – DIETI, Napoli, Italy

Stefan Plantikow Neo4j, Berlin, Germany

John Poelman IBM, New York, USA

Meikel Poess Server Technologies, Oracle Corporation, Redwood Shores, CA, USA

Nicolas Poggi Databricks Inc, Amsterdam, NL, BarcelonaTech (UPC), Barcelona, Spain

Artem Polyvyanyy The University of Melbourne, Parkville, VIC, Australia

Luigi Pontieri National Research Council, Institute for High Performance Computing and Networking (ICAR-CNR), Rende (CS), Italy

Arnaud Prat-Pérez Universitat Politècnica de Catalunya, Barcelona, Spain

Nuno Preguiça NOVA LINCS and DI, FCT, Universidade NOVA de Lisboa, Caparica, Portugal

Deepak Puthal Faculty of Engineering and Information Technologies, School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW, Australia

Jianzhong Qi The University of Melbourne, Melbourne, VIC, Australia

Yongrui Qin School of Computing and Engineering, University of Huddersfield, Huddersfield, UK

Christoph Quix Fraunhofer-Institute for Applied Information Technology FIT, Sankt Augustin, Germany

Do Le Quoc TU Dresden, Dresden, Germany

Francois Raab InfoSizing, Manitou Springs, CO, USA

Tilmann Rabl Database Systems and Information Management Group, Technische Universität Berlin, Berlin, Germany

Muhammad Rafi School of Computer Science, National University of Computer and Emerging Sciences, Karachi, Pakistan

Mohammad Saiedur Rahaman School of Science (Computer Science and IT), RMIT University, Melbourne, VIC, Australia

Erhard Rahm University of Leipzig, Leipzig, Germany

Vineeth Rakesh Data Mining and Machine Learning Lab, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Vijayshankar Raman IBM Research – Almaden, San Jose, CA, USA

Rajiv Ranjan School of Computing Science, Newcastle University, Newcastle upon Tyne, UK

Niranjan K. Ray Kalinga Institute of Industrial Technology (KIIT), Deemed to be University, Bhubaneswar, India

Manuel Resinas University of Seville, Sevilla, Spain

Juan Reutter Pontificia Universidad Católica de Chile, Santiago, Chile

Christian Rieger Data Management Lab, Reutlingen University, Reutlingen, Germany

Stefanie Rinderle-Ma Faculty of Computer Science, University of Vienna, Vienna, Austria

Norbert Ritter Department of Informatics, University of Hamburg, Hamburg, Germany

Nicoló Rivetti Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel

Rodrigo Rodrigues INESC-ID, IST (University of Lisbon), Lisbon, Portugal

Fábio Diniz Rossi Federal Institute of Education Science, and Technology Farroupilha (IFFAR), Alegrete, Brazil

Paolo Rosso eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Antonio Ruiz-Cortés University of Seville, Sevilla, Spain

Anisa Rula Department of Computer Science, Systems and Communication (DISCo), University of Milano-Bicocca, Milan, Italy

Mohammad Sadoghi University of California, Davis, CA, USA

S. Cenk Sahinalp Department of Computer Science, Indiana University Bloomington, Bloomington, IN, USA

Bibhudatta Sahoo National Institute of Technology Rourkela, Rourkela, India

Sampa Sahoo National Institute of Technology Rourkela, Rourkela, India

Sherif Sakr Institute of Computer Science, University of Tartu, Tartu, Estonia

Mohsen Amini Salehi High Performance Cloud Computing (HPCC) laboratory, University of Louisiana at Lafayette, Lafayette, Louisiana, USA

S. Salihoglu University of Waterloo, Waterloo, ON, Canada

Flora D. Salim School of Science (Computer Science and IT), RMIT University, Melbourne, VIC, Australia

Abhay Kumar Samal Trident Academy of Technology, Bhubaneswar, India

Donatello Santoro Dipartimento di Matematica, Informatica ed Economia, Università degli Studi della Basilicata, Potenza, Italy

Maribel Yasmina Santos Department of Information Systems, ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

Mohamed Sarwat School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Matthias J. Sax Confluent Inc, Palo Alto, CA, USA

K. Sayood Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

David L. Schmidt Hewlett Packard Enterprise, Houston, TX, USA

Armin O. Schmitt Department of Breeding Informatics, Georg-August University, Göttingen, Germany

Scott Schneider IBM Research AI, Yorktown Heights, NY, USA

Christian Schulz University of Vienna, Vienna, Austria

Arik Senderovich Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, Canada

Juan Sequeda Capsenta, Austin, TX, USA

Giuseppe Settanni Center for Digital Safety and Security, AIT Austrian Institute of Technology, Vienna, Austria

Furqan Shaikh School of Computer Science, National University of Computer and Emerging Sciences, Karachi, Pakistan

Siti Mariyam Shamsuddin UTM Big Data Centre, Universiti Teknologi Malaysia, Johor, Malaysia

Anil Shanbhag MIT, Cambridge, MA, USA

Marc Shapiro Sorbonne Université, LIP6 and INRIA, Paris, France

Shashi Shekhar Department of Computer Science, University of Minnesota, Minneapolis, MN, USA

Quan Z. Sheng Department of Computing, Macquarie University, Sydney, NSW, Australia

Richard Sidle IBM Research – Almaden, San Jose, CA, USA

Yogesh Simmhan Department of Computational and Data Sciences, Indian Institute of Science (IISc), Bangalore, India

Rainer Simon Austrian Institute of Technology, Vienna, Austria

Ramesh Sitaraman University of Massachusetts, Amherst, MA, USA

Florian Skopik Center for Digital Safety and Security, AIT Austrian Institute of Technology, Vienna, Austria

Alisa Smirnova Exascale Infolab, University of Fribourg, Fribourg, Switzerland

Mohamed A. Soliman Datometry, Inc, San Francisco, CA, USA

Andreas Solti Vienna University of Economics and Business, Vienna, Austria

Robert Soulé Università della Svizzera Italiana (USI), Lugano, Switzerland

Giancarlo Sperlí Department of Electrical Engineering and Information Technology, University of Naples “Federico II”, Naples, Italy

Satish Narayana Srirama Mobile and Cloud Lab, Institute of Computer Science, University of Tartu, Tartu, Estonia

George Stamoulis Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece

Darren Strash Department of Computer Science, Colgate University, Hamilton, NY, USA

Thorsten Strufe TU Dresden, Dresden, Germany

Torsten Suel Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Peng Sun School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

Yuhan Sun School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Tisinee Surapunt Beijing Institute of Technology, Beijing, China

Arun Suresh Microsoft, Washington, DC, USA

Willy Susilo Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia

Pierre Sutra Télécom Sud Paris, Évry, France

Anil Kumar Swain Kalinga Institute of Industrial Technology (KIIT), Deemed to be University, Bhubaneswar, India

Muhammad Atif Tahir School of Computer Science, National University of Computer and Emerging Sciences, Karachi, Pakistan

Lei Tang Clari Inc, Sunnyvale, CA, USA

Kanat Tangwongsan Mahidol University International College, Salaya, Thailand

Martin Theobald Université du Luxembourg, Luxembourg, Luxembourg

Christian Thomsen Department of Computer Science, Aalborg University, Aalborg, Denmark

Yuanyuan Tian IBM Research – Almaden, San Jose, CA, USA

Konstantin Todorov LIRMM, University of Montpellier, Montpellier, France

Marc Torrent-Moreno Data Science and Big Data Analytics Unit, EURECAT (Technology Centre of Catalonia), Barcelona, Spain

Juan Touriño Computer Architecture Group, Universidade da Coruña, A Coruña, Spain

Pinar Tözün IT University of Copenhagen, Copenhagen, Denmark

Guy Tremblay Département d’Informatique, Université du Québec à Montréal, Montréal, QC, Canada

Panayiotis Tsaparas Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

Ashok Kumar Turuk National Institute of Technology Rourkela, Rourkela, India

Jacopo Urbani Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Alejandro A. Vaisman Instituto Tecnológico de Buenos Aires (ITBA),
Buenos Aires, Argentina

Maurice Van Keulen Faculty of EEMCS, University of Twente, Enschede,
The Netherlands

Oskar van Rest Oracle, CA, USA

Dinusha Vatsalan Data61, CSIRO, Eveleigh, NSW, Australia

Research School of Computer Science, The Australian National University,
Acton, ACT, Australia

Jorge Veiga Computer Architecture Group, Universidade da Coruña,
A Coruña, Spain

Rossano Venturini Department of Computer Science, University of Pisa,
Pisa, Italy

Vassilios S. Verykios School of Science and Technology, Hellenic Open
University, Patras, Greece

Sebastiano Vigna Università degli Studi di Milano, Milano, Italy

Tobias Vinçon Data Management Lab, Reutlingen University, Reutlingen,
Germany

Hannes Voigt Dresden Database Systems Group, Technische Universität
Dresden, Dresden, Germany

Jóakim von Kistowski Institute of Computer Science, University of
Würzburg, Würzburg, Germany

Zacharias Voulgaris Data Science Partnership Ltd, London, UK

Thinkful LLC, New York, USA

Technics Publications LLC, Basking Ridge, NJ, USA

Lance A. Waller Department of Biostatistics and Bioinformatics, Rollins
School of Public Health, Emory University, Atlanta, GA, USA

Timo Walther Data Artisans GmbH, Berlin, Germany

Hua Wang Institute for Sustainable Industries and Liveable Cities, VU Re-
search, Victoria University, Melbourne, Australia

Siqi Wang School of Engineering and ICT, University of Tasmania, Hobart,
TAS, Australia

Shuliang Wang Beijing Institute of Technology, Beijing, China

Jochen De Weerdt KU Leuven, Leuven, Belgium

Matthias Weidlich Humboldt-Universität zu Berlin, Department of Com-
puter Science, Berlin, Germany

Thomas Weise Atrato Inc., San Francisco, CA, USA

Yonggang Wen School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

Lena Wiese Institute of Computer Science, Georg-August University, Göttingen, Germany

Peter T. Wood Birkbeck, University of London, London, UK

Yinghui Wu Washington State University, Pullman, WA, USA

Markus Wurzenberger Center for Digital Safety and Security, AIT Austrian Institute of Technology, Vienna, Austria

Marcin Wylot ODS, TU Berlin, Berlin, Germany

N. Yakovets Eindhoven University of Technology, Eindhoven, Netherlands

Da Yan Department of Computer Science, The University of Alabama at Birmingham, Birmingham, AL, USA

Robin Yancey Department of Electrical and Computer Engineering, University of California, Davis, CA, USA

Dingqi Yang eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Yuichi Yoshida National Institute of Informatics, Tokyo, Japan

Siti Sophiyati Yuhaniz Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia

Amrapali Zaveri Institute of Data Science, Maastricht University, Maastricht, The Netherlands

Hushairi Zen Universiti Malaysia Sarawak, Kota Samarahan, Sarawak, Malaysia

Kaiwen Zhang Department of Software and IT Engineering, École de technologie supérieure de Montréal, Université du Québec, Montréal, QC, Canada

Rui Zhang The University of Melbourne, Melbourne, VIC, Australia

Xuyun Zhang Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand

Peixiang Zhao Department of Computer Science, Florida State University, Tallahassee, FL, USA

Amelie Chi Zhou Shenzhen University, Shenzhen, China

Xiaofang Zhou School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD, Australia

Kaiyuan Zhu Department of Computer Science, Indiana University Bloomington, Bloomington, IN, USA

Roberto V. Zicari Frankfurt Big Data Lab, Goethe University Frankfurt, Frankfurt, Germany

Esteban Zimányi CoDE, Université Libre de Bruxelles, Brussels, Belgium

Nur Zincir-Heywood Dalhousie University, Halifax, NS, Canada

S. M. Zobaed High Performance Cloud Computing (HPCC) laboratory, University of Louisiana at Lafayette, Lafayette, LA, USA

Marcin Zukowski Snowflake Computing, San Mateo, CA, USA

A

Achieving Low Latency Transactions for Geo-replicated Storage with Blotter

Henrique Moniz¹, João Leitão²,
Ricardo J. Dias³, Johannes Gehrke⁴, Nuno
Preguiça⁵, and Rodrigo Rodrigues⁶

¹Google and NOVA LINCS, New York, USA

²DI/FCT/Universidade NOVA de Lisboa and
NOVA LINCS, Lisbon, Portugal

³SUSE Linux GmbH and NOVA LINCS,
Lisbon, Portugal

⁴Microsoft, Seattle, USA

⁵NOVA LINCS and DI, FCT, Universidade
NOVA de Lisboa, Caparica, Portugal

⁶INESC-ID, IST (University of Lisbon), Lisbon,
Portugal

Synonyms

Blotter design and protocols; Geo-replicated transactions; Non-monotonic Snapshot Isolation in Geo-replicated Systems

Definitions

Blotter is a protocol for executing transactions in geo-replicated storage systems with non-monotonic snapshot isolation semantics. A geo-replicated storage system is composed by a set of nodes running in multiple data centers

located in different geographical locations. The nodes in each data center replicate either all or a subset of the data items in the database, leading to a full replication or partial replication approach. Blotter was primarily designed for full replication scenarios but can also be used in partial replication scenarios. Under non-monotonic snapshot isolation semantics, a transaction reads from a snapshot that reflects all the writes from a set of transactions that includes, at least, all locally committed transactions and remote transactions known when the transaction starts. Two concurrent transactions conflict if their write set intersects, i.e., if there is a data item written by both. In such case, one of the transactions will abort.

Overview

Many Internet services are backed by geo-replicated storage systems in order to keep data close to the end user. This decision is supported by studies showing the negative impact of latency on user engagement and, by extension, revenue (Hoff 2009). While many of these systems rely on weak consistency for better performance and availability (DeCandia et al. 2007), there is also a class of applications that require support for strong consistency and transactions. For instance, many applications within Google are operating on top of Megastore (Baker et al. 2011), a system that provides ACID semantics within the same shard, instead of Bigtable (Chang et al.

2008), which provides better performance but weaker semantics. This trend also motivated the development of Spanner, which provides general serializable transactions (Corbett et al. 2012) and sparked other recent efforts in the area of strongly consistent geo-replication (Sovran et al. 2011; Saeida Ardekani et al. 2013a; Lloyd et al. 2013; Zhang et al. 2013; Kraska et al. 2013; Mahmoud et al. 2013).

In this chapter, we present Blotter, a transactional geo-replicated storage system, whose goal is to cut the latency penalty for ACID transactions in geo-replicated systems, by leveraging a recent isolation proposal called Non-Monotonic Snapshot Isolation (NMSI) (Saeida Ardekani et al. 2013a). We focus on the Blotter algorithms and discuss how they achieve: (1) at most one round-trip across data centers (assuming a fault-free run and that clients are proxies in the same data center as one of the replicas) and (2) read operations that are always served by the local data center.

To achieve these goals, Blotter combines a novel *concurrency control* algorithm that executes at the data center level, with a carefully configured Paxos-based (Lamport 1998) replicated state machine that replicates the execution of the concurrency control algorithm across data centers. Both of these components exploit several characteristics of NMSI to reduce the amount of coordination between replicas. In particular, the concurrency control algorithm leverages the fact that NMSI does not require a total order on the start and commit times of transactions. Such an ordering would require either synchronized clocks, which are difficult to implement, even using expensive hardware (Corbett et al. 2012), or synchronization between replicas that do not hold the objects accessed by a transaction (Saeida Ardekani et al. 2013b), which hinders scalability. In addition, NMSI allows us to use separate (concurrent) Paxos-based state machines for different objects, on which we geo-replicate the *commit* operation of the concurrency control protocol.

Compared to a previously proposed NMSI system called Jessy (Saeida Ardekani et al. 2013a), instead of assuming partial replication,

we target full replication, which is a common deployment scenario (Baker et al. 2011; Shute et al. 2013; Bronson et al. 2013). Our layering of Paxos on top of a concurrency control algorithm is akin to the Replicated Commit system, which layers Paxos on top of two-phase Locking (Mahmoud et al. 2013). However, by leveraging NMSI, we execute reads exclusively locally and run parallel instances of Paxos for different objects, instead of having a single instance per shard. Furthermore, when a client is either colocated with the Paxos leader or when that leader is in the closest data center to the client, Blotter can commit transactions within a single round-trip to the *closest* data center.

We have implemented Blotter as an extension to the well-known geo-replicated storage system Cassandra (Lakshman and Malik 2010). Our evaluation shows that, despite adding a small overhead in a single data center, Blotter performs much better than both Jessy and the protocols used by Spanner and also outperforms in many metrics a replication protocol that ensures snapshot isolation (Elnikety et al. 2005). This shows that Blotter can be a valid choice when several replicas are separated by high latency links, performance is critical, and the semantic differences between NMSI and snapshot isolation are acceptable by the application.

Key Research Findings

The research conducted to design and implement Blotter led to the following key contributions and findings:

1. A simple yet precise definition of non-monotonic snapshot isolation, a recent isolation model that aims at maximizing parallel execution of transactions in geo-replicated systems.
2. The design of Blotter, which combines a novel concurrency control protocol for a single data-center and its extension toward an arbitrary number of data centers by leveraging a carefully designed state machine replication

solution leveraging a variant of the Paxos algorithm.

3. The demonstration that it is possible to achieve strong consistency in geo-replicated storage systems within a modest latency envelop in fault-free runs.

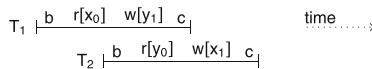
Non-monotonic Snapshot Isolation

We start by providing a specification of our target isolation level, NMSI, and discussing the advantages and drawbacks of this choice.

Snapshot Isolation Revisited

NMSI is an evolution of snapshot isolation (SI). Under SI, a transaction (logically) executes in a database snapshot taken at the transaction begin time, reflecting the writes of all transactions that committed before that instant. Reads and writes execute against this snapshot and, at commit time, a transaction can commit if there are no write-write conflicts with concurrent transactions. (In this context, two transactions are concurrent if the intervals between their begin and commit times overlap.)

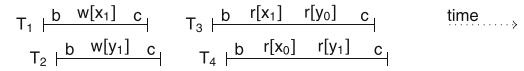
Snapshot isolation exhibits the write-skew anomaly, where two concurrent transactions T_1 and T_2 start by reading x_0 and y_0 , respectively, and later write y_1 and x_1 , as exemplified in the following figure. This execution is admissible under snapshot isolation, as there is no write-write conflict, and each transaction has read from a database snapshot. However, this execution is not serializable.



Specification of NMSI

NMSI weakens the SI specification in two ways. First, the snapshots against which transactions execute do not have to reflect the writes of a monotonically growing set of transactions. In other words, it is possible to observe what is called a “long fork” anomaly, where there can exist two concurrent transactions t_a and t_b that commit, writing to different objects, and two other transactions that start subsequently, where

one sees the effects of t_a but not t_b , and the other sees the effects of t_b but not t_a . The next figure exemplifies an execution that is admissible under NMSI but not under SI, since under SI both T_3 and T_4 would see the effects of both T_1 and T_2 because they started after the commit of T_1 and T_2 .



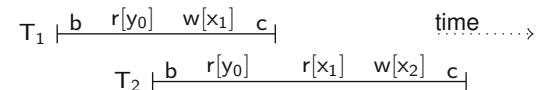
Second, instead of forcing the snapshot to reflect a subset of the transactions that committed at the transaction begin time, NMSI gives the implementation the flexibility to reflect a more convenient set of transactions in the snapshot, possibly including transactions that committed *after* the transaction began. This property, also enabled by serializability, is called *forward freshness* (Saeida Ardekani et al. 2013a).

Definition 1 (Non-Mon. Snapshot Isol. (NMSI))

An implementation of a transactional system obeys NMSI if, for any trace of the system execution, there exists a partial order \prec among transactions that obeys the following rules, for any pair of transactions t_i and t_j in the trace:

1. if t_j reads a value for object x written by t_i then $t_i \prec t_j \wedge \nexists t_k$ writing to $x : t_i \prec t_k \prec t_j$
2. if t_i and t_j write to the same object x then either $t_i \prec t_j$ or $t_j \prec t_i$.

The example in Fig. 1 obeys NMSI but not SI, as the depicted partial order meets Definition 1.



$$T_0 \prec T_1 \prec T_2$$

Achieving Low Latency Transactions for Geo-replicated Storage with Blotter, Fig. 1 Example execution obeying NMSI but not SI. This assumes the existence of a transaction T_0 that writes the initial values for x and y

The Benefits of NMSI in Replicated Settings

NMSI weakens the specification of SI in a way that can be leveraged for improving performance in replicated settings.

The possibility of having “long forks” allows, in a replicated setting, for a single (local) replica to make a decision concerning what data the snapshot should read. This is because it is not necessary to enforce a serialization between all transaction begin and commit operations, although it is still necessary to check for write-write conflicts,

In the case of “forward freshness,” this allows for a transaction to read (in most cases) the most recent version of a data object at a given replica, independently of the instant when the transaction began. This not only avoids the bookkeeping associated with tracking transaction start times but also avoids a conflict with transactions that might have committed after the transaction began.

Impact of NMSI for Applications

We analyze in turn the impact of “forward freshness” and “long forks” for application developers. Forward freshness allows a transaction t_a to observe the effects of another transaction t_b that committed after t_a began (in real time). In this case, the programmer must decide whether this is a violation of the intended application semantics, which is analogous to decide if serializability or strict serializability is the most adequate isolation level for a given application. Long forks allow two transactions to execute against different branches of a forked database state, provided there are no write-write conflicts. In practice, the main implication of this fact is that the updates made by users may not become instantly visible across all replicas. For example, this could cause two users of a social network to each think that they were the first to post a new promotion on their own wall, since they do not see each other’s posts immediately (Sovran et al. 2011). Again, the programmer must reason whether this is admissible. In this case, a mitigating factor is that this anomaly does not cause the consistency of the database to break. (This is in contrast with the “write-skew” anomaly, which is present in both SI and NMSI.) Furthermore, in the

particular case of our protocol, the occurrence of anomalies is very rare: for a “long fork” to occur, two transactions must commit in two different data centers, form a quorum with a third data center, and both complete before hearing from the other.

Finally, NMSI allows consecutive transactions from the same client to observe a state that reflects a set of transactions that does not grow monotonically (when consecutive transactions switch between two different branches of a long fork). However, in our algorithms, this is an unlikely occurrence, since it requires that a client connects to different data centers in a very short time span.

Blotter Protocols

We now describe the Blotter protocols, introducing first the concurrency protocol algorithm in a single data center scenario, and then how this protocol can be replicated to multiple data centers. A more detailed explanation of the protocols including their evaluation can be found elsewhere (Moniz et al. 2017).

System Model

Blotter is designed to run on top of any distributed storage system with nodes spread across one or multiple data centers. We assume that each data object is replicated at all data centers. Within each data center, data objects are replicated and partitioned across several nodes. We make no restrictions on how this intra-data center replication and partitioning takes place. We assume that nodes may fail by crashing and recover from such faults. When a node crashes, it loses its volatile state, but all data that was written to stable storage is accessible after recovery.

We use an asynchronous system model, i.e., we do not assume any known bounds on computation and communication delays. We do not prescribe a fixed bound on the number of faulty nodes within each data center – this depends on the intra-data center replication protocol used.

Blotter Architecture

The client library of Blotter exposes an API with the expected operations: begin a new

transaction, *read* an object given its identifier, *write* an object given its identifier and new value, and *commit* a transaction, which either returns commit or abort.

The set of protocols that comprise Blotter are organized into three different components:

Blotter intra-data center replication. At the lowest level, we run an intra-data center replication protocol to mask the unreliability of individual machines within each data center. This level must provide the protocols above it with the vision of a single logical copy (per data center) of each data object and associated metadata, which remains available despite individual node crashes. We do not prescribe a specific protocol for this layer, since any of the existing protocols that meet this specification can be used.

Blotter Concurrency Control. These are the protocols that ensure transaction atomicity and NMSI isolation in a single data center and, at the same time, are extensible to multiple data centers by serializing a single protocol step.

Inter-data Center Replication. This completes the protocol stack by replicating a subset of the steps of the concurrency control protocol across data centers. It implements state machine replication (Schneider 1990; Lamport 1978) by judiciously applying Paxos (Lamport 1998) to the concurrency control protocol to avoid unnecessary coordination across data centers.

Single Data Center Protocol

The single data center concurrency control module consists of the following three components: the client library and the transaction managers (TM), which are non-replicated components that act as a front end providing the system interface and implementing the client side of the transaction processing protocol, respectively; and the data managers (DM), which are the replicated components that manage the information associated with data objects.

Client Library. This provides the interface of Blotter, namely, *begin*, *read*, *write*, and *commit*.

The *begin* and *write* operations are local to the client. *Read* operations are relayed to the TM, who returns the values and metadata for the objects that were read. The written values are buffered by the client library and only sent to the TM at *commit* time, together with the accumulated metadata for the objects that were read. This metadata is used to set the versions that running transactions must access, as explained next.

Transaction Manager (TM). The TM handles the two operations received from the clients: *read* and *commit*. For *reads*, it merely relays the request and reply to or from the data manager (DM) responsible for the object being read. Upon receiving a *commit* request, the TM acts as a coordinator of a two-phase commit (2PC) protocol to enforce the all-or-nothing atomicity property. The first phase sends a *dm-prewrite-request*, with the newly written values, to all DMs storing written objects. Each DM verifies if the write complies with NMSI. If none of the DMs identifies a violation, the TM sends the DMs a *dm-write* message containing the metadata with snapshot information aggregated from all replies on the first phase; otherwise, it sends a *dm-abort*.

Data Manager (DM). The core of the concurrency control logic is implemented by the DM, which maintains the data and meta-data necessary to provide NMSI. Algorithm 1 presents the handlers for the three types of requests. Now, we explain how these functions enforce NMSI rules presented in our definition.

Partial order \prec . We use a multi-version protocol, i.e., the system maintains a list of versions for each object. This list is indexed by an integer version number, which is incremented every time a new version of the object is created (e.g., for a given object x , overwriting x_0 creates version x_1 , and so on). In a multi-versioned storage, the \prec relation can be defined by the version number that transactions access, namely, if t_i writes x_m and t_j writes x_n , then $t_i \prec t_j \Leftrightarrow m < n$; and if t_i writes x_m and t_j reads x_n , then $t_i \prec t_j \Leftrightarrow m \leq n$.

Algorithm 1: Single data center DM protocols

```

    // read operation
1  upon { dm-read, T, x } from TM do
2      processRead { T, x, TM };

    // prewrite operation
3  upon { dm-prewrite, T, x, value } from TM do
4      if x.prewrite ≠ ⊥ then
5          // another prewrite is pending
6          x.pending ← x.pending ∪ {(T, x, value, TM)};
7      else
8          processPrewrite { T, x, value, TM };

    // write operation
9  upon { dm-write, T, x, agg-startd-before } from TM do
10     for each T' in agg-startd-before do
11         if T' not in x.snapshot then
12             x.snapshot[T'] ← x.last;
13
14         x.last ← x.last + 1;
15         x.value[x.last] ← x.nextvalue;
16         finishWrite { T, x, TM };

    // abort operation
17  upon { dm-abort, T, x } from TM do
18      finishWrite { T, x, TM };

    // process read operation
19  processRead { T, x, TM }
20      if T ∉ x.snapshot then
21          if x.prewrite ≠ ⊥ then
22              x.buffered ← x.buffered ∪ {(T, TM)};
23              return
24          else
25              x.snapshot[T] ← x.last;

26      version ← x.snapshot[T];
27      value ← x.value[version];
28      send { read-response, T, value, {T'|x.snapshot[T'] < version} } to TM;

    // process dm-prewrite request
29  processPrewrite { T, x, value, TM }
30      if x.snapshot[T] ≠ ⊥ ∧ x.snapshot[T] < x.last then
31          // there is a write-write conflict
32          send { prewrite-response, reject, ⊥ } to TM;
33      else
34          x.prewrite ← T;
35          x.nextvalue ← value;
36          send { prewrite-response, accept, {T'|T' ∈ x.snapshot} } to TM;

    // clean prewrite information and serve buffered reads and pending prewrites
37  finishWrite { T, x, TM }
38      if x.prewrite = T then
39          x.nextvalue ← ⊥; x.prewrite ← ⊥;
40
41      for each (T, TM) in x.buffered do
42          processRead { T, x, TM };
43
44      if x.pending ≠ ⊥ then
45          (T, x, value, TM) ← removeFirst(x.pending);
46          processPrewrite { T, x, value, TM };

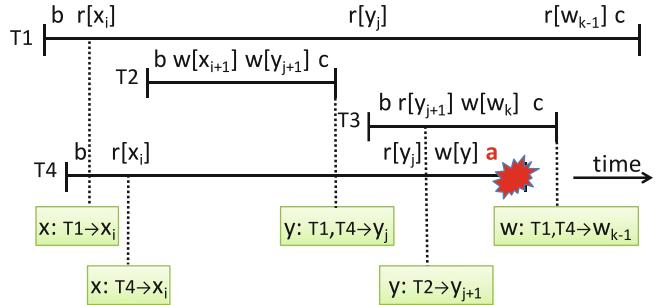
```

NMSI rule number 1. Rule number 1 of the definition of NMSI says that, for object x , transaction t must read the value written by the “latest” transaction that updated x (according to \prec). To illustrate this, consider the example run in Fig. 2. When a transaction T_1 issues its first read operation, it can read the most recently committed version of the object, say x_i written by T_0 (leading to $T_0 \prec T$). If, subsequently, some other

transaction T_2 writes x_{i+1} ($T_0 \prec T_2$), then the protocol must prevent T_1 from either reading or overwriting the values written by T_2 . Otherwise, we would have $T_0 \prec T_2 \prec T_1$ and T_1 should have read the value for object x written by T_2 (i.e., x_{i+1}) instead of that written by T_0 (i.e., x_i). Next, we detail how this is achieved first for reads, then writes, and then how to enforce the rule transitively.

Achieving Low Latency Transactions for Geo-replicated Storage with Blotter, Fig. 2

Example run



Reading the latest preceding version. The key to enforcing this requirement is to maintain state associated with each object, stating the version a running transaction must read, in case such a restriction exists. In the previous example, if T_2 writes x_{i+1} , this state records that T_1 must read x_i .

To achieve this, our algorithm maintains a per-object dictionary data structure ($x.snapshot$), mapping the identifier of a transaction t to a particular version of x that t either must read or has read from. Figure 2 depicts the changes to this data structure in the shaded boxes at the bottom of the figure. When the DM processes a read of x for t (function `processRead`), if the dictionary has no information for t , the most recent version is read, and this information is stored in the dictionary. Otherwise, the specified version is returned.

In the previous example, T_1 must record the version it reads in the $x.snapshot$ variable. Subsequently, when the commit of T_2 overwrites that version of x , we are establishing that $T_2 \not\propto T_1$. As such, if T_2 writes to another object y , creating y_{j+1} , then it must also force T_1 to read the preceding version y_j . To do this, when transaction T_2 commits, for every transaction t that read (or must read) an older version of object x (i.e., the transactions with entries in the dictionary of x), the protocol will store in the dictionary of every other object y written by T_2 that t must read the previous version of y , unless an even older version is already prescribed (`dm-write` handler). In this particular example, $y.snapshot$ would record that T_1 and T_4 must read version y_j , since, at commit time, $x.snapshot$ indicates that these transactions read x_i .

Preventing illegal overwrites. In the previous example, we must also guarantee that T_1 does not overwrite any value written by T_2 . To enforce this, it suffices to verify, at the time of the commit of transaction t , for every object written by t , if T should read its most recent version. If this is the case, then the transaction can commit, since no version will be incorrectly overwritten; otherwise, it must abort (function `processPrewrite`). In the example, T_4 aborts, since $y.snapshot$ records that T_4 must read y_j and a more recent version exists (y_{j+1}). Allowing T_4 to commit and overwrite y_{j+1} would lead to $T_2 \prec T_4$. This breaks rule number 1 of NMSI, since it would have required T_1 to read x_{i+1} written by T_2 , which did not occur.

Applying the rules transitively. Finally, for enforcing rule number 1 of the definition of NMSI in a transitive manner, it is also necessary to guarantee the following: if T_2 writes x_{i+1} and y_{j+1} , and subsequently another transaction T_3 reads y_{j+1} and writes w_k , then the protocol must also prevent T_1 from reading or overwriting the values written by T_3 , otherwise we would have $T_0 \prec T_2 \prec T_3 \prec T_1$, and thus T_1 should also have read x_{i+1} .

To achieve this, when transaction T_3 (which read y_{j+1}) commits, for every transaction t that must read version y_l with $l < j + 1$ (i.e., the transactions that had entries in the dictionary of y when t_o read y), the protocol will store in the dictionary of every other object w written by T_3 that t must read the previous version of w (if an older version is not already specified). In the example, since the state for $y.snapshot$ after T_2 commits specifies that T_1 must read version y_j , then, when T_3 commits, $w.snapshot$ is updated

to state that T_1 and T_4 must read version w_{k-1} . This is implemented by collecting dependencies in read and pre-write functions and use these information to update the snapshot information for all objects in the write function.

NMSI rule number 2. Rule number 2 of the NMSI definition says that any pair of transactions that write the same object x must have a relative order, i.e., either $t_i \prec t_j$ or $t_j \prec t_i$. This order is defined by the version number of x created by each transaction.

Therefore, it remains to ensure that this is a partial order (i.e., no cycles). A cycle could appear if two or more transactions concurrently committed a chain of objects in a different order, e.g., if t_m wrote both x_i and y_{j+1} and t_n wrote both x_{i+1} and y_j . To prevent this, it suffices to use a two-phase commit protocol where, for each object, a single accepted prepare can be outstanding at any time.

Geo-Replication

Blotter implements geo-replication, with each object replicated in all data centers, using Paxos-based state machine replication (Lamport 1998; Schneider 1990). In this model, all replicas execute a set of client-issued commands according to a total order, thus following the same sequence of states and producing the same sequence of responses. We view each data center as a state machine replica. The state is composed by the database (i.e., all data objects and associated metadata), and the state machine commands are the `tm-read` and the `tm-commit` of the TM-DM interface.

Despite being correct, this approach has three drawbacks, which we address in detail in a separate paper (Moniz et al. 2017).

First, read operations in our concurrency control protocol are state machine commands that mutate the state, thus requiring an expensive consensus round. To address this, we leverage the fact that the NMSI properties allow for removing this information from the state machine, since the modified state only needs to be used locally.

Second, the total order of the state machine precludes the concurrent execution of two com-

mits, even for transactions that do not conflict. In this case, we leverage the fact that the partial order required by the NMSI definition can be built by serializing the `dm-prewrite` operation on a per-object basis, instead of serializing `tm-commits` across all objects. As such, instead of having one large state machine whose state is defined by the entire database, we can have one state machine per object, with the state being the object (including its metadata), and supporting only the `dm-prewrite` operation.

Finally, each Paxos-based state machine command requires several cross-data center message delays (depending on the variant of Paxos used). We adjusted the configuration of Paxos to reduce the cross data center steps to a single round-trip (from the client of the protocol, i.e., the TM) for update transactions, by using Multi-Paxos (Lamport 1998) and configuring Paxos to only tolerate one unplanned outage of a data center. In fact, this assumption is common in existing deployed systems (Ananthanarayanan et al. 2013; Corbett et al. 2012). (Planned outages are handled by re-configuring the Paxos membership Lamport et al. 2010.)

Examples of Applications

Blotter can be leveraged to design multiple large-scale geo-distributed applications, and in particular web applications whose semantics are compatible with the guarantees (and anomalies) associated with NMSI.

In particular, we have explored how to implement a set of the main operations of a microblogging platform, such as Twitter.

In our implementation of a simplistic version of Twitter, we model the timeline (sometimes called wall in the context of social network applications) as a data object in the (geo-replicated) data storage service. Furthermore, we associate with each user a set of followers and a set of followees. Each of these sets is modeled as an independent data object in the data storage.

We have supported three different user interactions. We selected these operations by relying on the model presented in Zhang et al. (2013):

- *Post-tweet* appends a tweet to the timeline of a user and its followers, which results in a transaction with many reads and writes, since one has to read the followers of the user posting the tweet and then write to the timeline of both the user and each corresponding follower.
- *Follow-user* appends new information in the set of followers to the profiles of the follower and the followee, which results in a transaction with two reads and two writes.
- *Read-timeline* reads the wall of the user issuing the operation, resulting in a single read operation.

In this case, the long fork anomaly can only result in a user observing (slightly) stale data, and perhaps even updating its own state based on that data, e.g., a user that decides to follow another can miss a few entries of that new user in his own timeline. Note that reading stale data can be addressed by waiting for the information about transactions to be propagated across all data centers in the deployment.

We also implemented the RUBiS benchmark, which models an auction site similar to eBay, on top of Blotter.

We ported the benchmark from using a relational database as the storage back end to using a key-value store. Each row of the relational database is stored with a key formed by the name of the table and the value of the primary key. We additionally store data for supporting efficient queries (namely, indexes and foreign keys).

In this benchmark, users issue operations such as selling, browsing, bidding, or buying items and consulting a personal profile that lists outstanding auctions and bids. All of these operations were implemented as Blotter transactions. In this application, the long fork anomaly that is allowed by NMSI has a few implications. In particular, when browsing the items that are available to be sold (or in auction), users can observe stale data, in the form of missing a few objects that are relevant for their queries.

Surprisingly, since bidding and buying an item involves writing to a single object in the data store in our adaptation, the long fork anomaly does

not affect these semantics, since all transactions that write on this object are totally ordered by the Paxos leader that is responsible for mediating the access to it.

The experimental results that evaluate the performance of these implementations can be in a separate paper (Moniz et al. 2017).

Future Directions of Research

In this chapter, we have presented the design of Blotter, a novel system architecture and set of protocols that exploits the benefits of NMSI to improve the performance of geo-replicated transactional systems.

Blotter demonstrates the practical benefits that can be achieved by relaxing the well-known and widely used SI model. To better frame the applications that can benefit from the Blotter design, we have presented a simple yet precise specification of NMSI and discussed how this specification differs in practice from the SI specification.

An interesting direction for future research in this context is to explore automatic mechanisms to check if an existing application designed for the SI model could safely be adapted to operate under NMSI. This could be achieved by exploiting application state invariants, and modeling transactions through their preconditions and post-conditions.

Acknowledgements Computing resources for this work were provided by an AWS in Education Research Grant. The research of R. Rodrigues is funded by the European Research Council (ERC-2012-StG-307732) and by FCT (UID/CEC/50021/2013). This work was partially supported by NOVA LINCS (UID/CEC/04516/2013) and EU H2020 LightKone project (732505). This chapter is derived from Moniz et al. (2017).

References

- Ananthanarayanan R, Basker V, Das S, Gupta A, Jiang H, Qiu T, Reznichenko A, Ryabkov D, Singh M, Venkataraman S (2013) Photon: fault-tolerant and scalable joining of continuous data streams. In: SIGMOD'13: proceeding of 2013 international conference on management of data, pp 577–588

- Baker J, Bond C, Corbett JC, Furman J, Khorlin A, Larson J, Leon JM, Li Y, Lloyd A, Yushprakh V (2011) Megastore: providing scalable, highly available storage for interactive services. In: Proceeding of the conference on innovative data system research (CIDR), pp 223–234. http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf
- Bronson et al N (2013) Tao: facebook’s distributed data store for the social graph. In: Proceeding of the 2013 USENIX annual technical conference, pp 49–60
- Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):4:1–4:26. <http://doi.acm.org/10.1145/1365815.1365816>
- Corbett et al JC (2012) Spanner: Google’s globally-distributed database. In: Proceeding of the 10th USENIX conference on operating systems design and implementation, OSDI’12, pp 251–264. <http://dl.acm.org/citation.cfm?id=2387880.2387905>
- DeCandia et al G (2007) Dynamo: Amazon’s highly available key-value store. In: Proceeding of the 21st ACM symposium on operating systems principles, pp 205–220. <http://doi.acm.org/10.1145/1294261.1294281>
- Elnikety S, Zwaenepoel W, Pedone F (2005) Database replication using generalized snapshot isolation. In: Proceedings of the 24th IEEE symposium on reliable distributed systems, SRDS’05. IEEE Computer Society, Washington, DC, pp 73–84. <https://doi.org/10.1109/RELDIS.2005.14>
- Hoff T (2009) Latency is everywhere and it costs you sales – how to crush it. Post at the high scalability blog. <http://tinyurl.com/5g8mp2>
- Kraska T, Pang G, Franklin MJ, Madden S, Fekete A (2013) MDCC: multi-data center consistency. In: Proceeding of the 8th ACM European conference on computer systems, EuroSys’13, pp 113–126. <http://doi.acm.org/10.1145/2465351.2465363>
- Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. SIGOPS Oper Syst Rev 44(2):35–40. <http://doi.acm.org/10.1145/1773912.1773922>
- Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. <http://doi.acm.org/10.1145/359545.359563>
- Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169. <http://doi.acm.org/10.1145/279227.279229>
- Lamport L, Malkhi D, Zhou L (2010) Reconfiguring a state machine. ACM SIGACT News 41(1):63–73
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Proceeding of the 10th USENIX conference on networked systems design and implementation, NSDI’13, pp 313–328. <http://dl.acm.org/citation.cfm?id=2482626.2482657>
- Mahmoud H, Nawab F, Pucher A, Agrawal D, El Abbadi A (2013) Low-latency multi-datacenter databases using replicated commit. Proc VLDB Endow 6(9):661–672. <http://dl.acm.org/citation.cfm?id=2536360.2536366>
- Moniz H, Leitão J, Dias RJ, Gehrke J, Preguiça N, Rodrigues R (2017) Blotter: low latency transactions for geo-replicated storage. In: Proceedings of the 26th international conference on World Wide Web, International World Wide Web conferences steering committee, WWW ’17, Perth, pp 263–272. <https://doi.org/10.1145/3038912.3052603>
- Saeida Ardekani M, Sutra P, Shapiro M (2013a) Non-monotonic snapshot isolation: scalable and strong consistency for geo-replicated transactional systems. In: Proceeding of the 32nd IEEE symposium on reliable distributed systems (SRDS 2013), pp 163–172. <https://doi.org/10.1109/SRDS.2013.25>
- Saeida Ardekani M, Sutra P, Shapiro M, Preguiça N (2013b) On the scalability of snapshot isolation. In: Euro-Par 2013 parallel processing. LNCS, vol 8097. Springer, pp 369–381. https://doi.org/10.1007/978-3-642-40047-6_39
- Schneider FB (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv 22(4):299–319. <http://doi.acm.org/10.1145/98163.98167>
- Shute J, Vingralek R, Samwel B, Handy B, Whipkey C, Rollins E, Oancea M, Littlefield K, Menestrina D, Ellner S, Cieslewicz J, Rae I, Stancescu T, Apte H (2013) F1: a distributed SQL database that scales. Proc VLDB Endow 6(11):1068–1079. <https://doi.org/10.14778/2536222.2536232>
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceeding of the 23rd ACM symposium on operating systems principles, SOSP’11, pp 385–400. <http://doi.acm.org/10.1145/2043556.2043592>
- Zhang Y, Power R, Zhou S, Sovran Y, Aguilera M, Li J (2013) Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In: Proceedings of the 24th ACM symposium on operating systems principles, SOSP, pp 276–291. <http://doi.acm.org/10.1145/2517349.2522729>

ACID Properties in MMOGs

- ▶ Transactions in Massively Multiplayer Online Games

Active Disk

- ▶ Active Storage

Active Storage

Ilia Petrov¹, Tobias Vinçon¹, Andreas Koch², Julian Oppermann², Sergey Hardock³, and Christian Rieger¹

¹Data Management Lab, Reutlingen University, Reutlingen, Germany

²Embedded Systems and Applications Group, TU Darmstadt, Darmstadt, Germany

³Databases and Distributed Systems Group, TU Darmstadt, Darmstadt, Germany

Synonyms

Active disk; Intelligent disk; Near-Data Processing; Programmable Memory/Processing In Memory (PIM)

Overview

In brief, *Active Storage* refers to an architectural hardware and software paradigm, based on co-location storage and compute units. Ideally, it will allow to execute application-defined data- or compute-intensive operations in situ, i.e., within (or close to) the physical data storage. Thus *Active Storage* seeks to minimize expensive data movement, improving performance, scalability, and resource efficiency. The effective use of *Active Storage* mandates new architectures, algorithms, interfaces, and development toolchains.

Over the last decade, we are witnessing a clear trend toward the fusion of the compute-intensive and the data-intensive paradigms on architectural, system, and application level. On the one hand, large computational tasks (e.g., simulations) tend to feed growing amounts of data into their complex computational models; on the other hand, database applications execute computationally intensive ML and analytics-style workloads on increasingly large data sets. Both result in massive *data transfers* across the memory hierarchy, which block the CPU, causing unnecessary CPU waits and thus impair performance, scalability, and resource efficiency. The root cause

for this phenomenon lies in the generally low data locality as well as in traditional architectures and algorithms, which operate on the *data-to-code* principle. It requires data and program code to be transferred to the processing elements to be executed. Although *data-to-code* simplifies development and system architectures, it is inherently bounded by the *von Neumann bottleneck*.

These trends are impacted by the following recent developments: (a) *Moore's law* is said to be cooling down for different types of semiconductor elements, and *Dennard scaling* is coming to an end. The latter postulates that performance per watt grows at approximately the rate mandated by Moore's law. (Besides the scalability of cache coherence protocols, *Dennard scaling* is among the frequently quoted reasons as to why modern many-core CPUs do not have the 128 cores that would otherwise be technically possible by now – see also Muramatsu et al. (2004) and Hardavellas et al. (2011)) As a result compute performance improvements cannot be based on the expectation of increasing clock frequencies and therefore mandate changes in the hardware and software architectures. (b) Modern systems can offer much *higher levels of parallelism*, yet scalability and the effective use of parallelism are limited by the programming models as well as by amount and type of data transfers. (c) *Access gap* and Memory Wall storage (DRAM, Flash, HDD) is getting larger and cheaper; however access latencies decrease at much lower rates. This trend also contributes to slow data transfers and to blocking processing at the CPU. (d) Modern data sets are large in volume (machine data, scientific data, text) and are growing fast (Szalay and Gray 2006). (e) Modern workloads (hybrid/HTAP or analytics-based such as OLAP or ML) tend to have low data locality and incur large scans (sometimes iterative) that result in massive data transfers.

In essence, due to system architectures and processing principles, current workloads require transferring growing volumes of large data through the virtual memory hierarchy, from the physical storage location to the

processing elements, which limits performance and scalability and worsens resource and energy efficiency.

Nowadays, three important technological developments open an opportunity to counter these drawbacks. Firstly, hardware manufactures are able to *fabricate combinations of storage and compute elements at reasonable costs* and package them within the same device. Secondly, the fact that this trend covers virtually all levels of the memory hierarchy: (a) CPU and caches, (b) memory and compute, (c) storage and compute, (d) accelerators – specialized CPUs and storage, and eventually (e) network and compute. Thirdly, as magnetic/mechanical storage is being replaced with semiconductor nonvolatile technologies (Flash, Non-Volatile Memories– NVM), another key trend emerges: the device internal bandwidth, parallelism, and access latencies are significantly better than the external ones (device-to-host). This is due to various reasons: interfaces, interconnect, physical design, and architectures.

Active Storage is a concept that targets the execution data processing operations (fully or partially) *in situ*: within the compute elements on the respective level of the storage hierarchy, close to where data is physically stored and transfer the results back, without moving the raw data. The underlying assumptions are: (a) the result size is much smaller than the raw data, hence less frequent and smaller-sized data transfers; or (b) the *in situ* computation is faster and more efficient than on the host, thus higher performance and scalability or better efficiency. Related concepts are *in situ processing*, *In-Storage Processing*, *smart storage*, and *Near-Data Processing* (Balasubramonian et al. 2014). The *Active Storage* paradigms have profound impact on multiple aspects:

1. *Interfaces*: hardware and software interfaces need to be extended, and new abstractions need to be introduced. This includes device and storage interfaces and operating systems and I/O abstractions: operations and conditions, records/objects vs. blocks, atomic primitives, and transactional support.

2. *Heterogeneity* in terms of storage and computer hardware interfaces is to be addressed.
3. *Toolchain*: extensive tool support is necessary to utilize Active Storage: compilers, hardware generators, debugging and monitoring tools, and advisors.
4. *Placement* of data and computation across the hierarchy is crucial to efficiency.
5. *Workload adaptivity* is a major goal as static assignments lower the placement and collocation effects.

These challenges already attract research focus, as todays accelerators exhibit Active Storage alike characteristics in a simplistic manner, i.e., GPUs and FPGAs are defined by a considerably higher level of internal parallelism and bandwidth in contrast to their connection to the host system. Specialized computation already uses hardware programmable with high-level synthesis toolchains like TaPaSCo (Korinth et al. 2015) and co-location with storage elements and often necessitate for shared virtual memory. Classical research questions, e.g., about a dynamic workload distribution, data dependence, and flexible data placement are approached by Fan et al. (2016), Hsieh et al. (2016), and Chen and Chen (2012), respectively. The significant potential arising with Near-Data Processing is investigated under perfect conditions by Kotra et al. (2017) stating performance boosts of about 75%.

Key Research Findings

Storage

The concept of *Active Storage* is not new. Historically it is deeply rooted in the concept of *database machines* (DeWitt and Gray 1992; Boral and DeWitt 1983) developed in the 1970s and 1980s. Boral and DeWitt (1983) discusses approaches such as processor-per-track or processor-per-head as an early attempt to combine storage and simple computing elements to accelerate data processing. Existing I/O bandwidth and parallelism are claimed to be the limiting factor to justify parallel DBMS. While this conclusion is not surprising given the characteristics of magnetic/mechanical storage

combined with Amdahl's balanced systems law, it is revised with modern technologies. Modern semiconductor storage technologies (NVM, Flash) are offering high raw bandwidth and levels of parallelism. Boral and DeWitt (1983) also raises the issue of temporal locality in database applications, which has been questioned back then and is considered to be low in modern workloads, causing unnecessary data transfers. Near-Data Processing and Active Storage present an opportunity to address it.

The concept of *Active Disk* emerged toward the end of the 1990s and early 2000s. It is most prominently represented by systems such as Active Disk (Acharya et al. 1998), IDISK (Keeton et al. 1998), and Active Storage/Disk (Riedel et al. 1998). While database machines attempted to execute fixed primitive access operations, *Active Disk* targets executing application-specific code on the drive. Active Storage/Disk (Riedel et al. 1998) relies on processor-per-disk architecture. It yields significant performance benefits for I/O-bound scans in terms of bandwidth, parallelism, and reduction of data transfers. IDISK (Keeton et al. 1998) assumed a higher complexity of data processing operations compared to Riedel et al. (1998) and targeted mainly analytical workloads and business intelligence and DSS systems. Active Disk (Acharya et al. 1998) targets an architecture based on on-device processors and pushdown of custom data processing operations. Acharya et al. (1998) focusses on programming models and explores a streaming programming model, expressing data-intensive operations as so-called disklets, which are pushed down and executed on the disk processor.

An extension of the above ideas (Sivathanu et al. 2005) investigates executing operations on the RAID controller. Yet, classical RAID technologies rely on general-purpose CPUs that operate well with slow mechanical HDDs, are easily overloaded, and turn into a bottleneck with modern storage technologies (Petrov et al. 2010).

Although in the Active Disk, concept increases the scope and applicability, it is equally impacted by bandwidth limitations and high manufacturing costs. Nowadays two

trends have important impact. On the one hand, semiconductor storage technologies (NVM, Flash) offer significantly higher bandwidths, lower latencies, and levels of parallelism. On the other hand, hardware vendors are able to fabricate economically combinations of storage and compute units and package them on storage devices. Both combined the result in new generation of Active Storage devices.

Smart SSDs (Do et al. 2013) or multi-stream SSDs aim to achieve better data processing performance by utilizing on device resources and pushing down data processing operations close to the data. Programming models such as *SSDlets* are being proposed. One trend is *In-Storage Processing* (Jo et al. 2016; Kim et al. 2016) that presents significant performance increase on embedded CPUs for standard DBMS operators. Combinations of storage and GPGPUs demonstrate an increase of up to 40x (Cho et al. 2013a). IBEX (Woods et al. 2013, 2014) is a system demonstrating operator pushdown on FPGA-based storage.

Do et al. (2013) is one of the first works to explore offloading parts of data processing on Smart SSDs, indicating potential of significant performance improvements (up to 2.7x) and energy savings (up to 3x). Do et al. (2013) defines a new session-based communication protocol (DBMS-SmartSSD) comprising three operations: OPEN, CLOSE, and GET. In addition they define a set of APIs for on-device functionality: Command API, Thread API, Data API, and Memory API. It does not only enable pushdown but also workload-dependent, cooperative processing. In addition, Do et al. Do et al. (2013) identify two research questions: (i) How can *Active Storage* handle the problem of on-device processing at the presence of a more recent version of the data in the buffer? (ii) What is the efficiency of operation pushdown at the presence of large main memories? The latter becomes obvious in the context of large data sets (Big Data) and computationally intensive operations.

Similarly, Seshadri et al. (2014) propose and describe a user-programmable SSD called Willow, which allows the users to augment the storage device with the application-specific logic.

In order to provide the new functionality of the SSD for a certain application, three subsystems must be appropriately modified to support a new set of RPC commands: the application, the kernel driver, and the operating system running on each storage processing unit inside the Flash SSD.

The initial ideas of Do et al. (2013) have been recently extended in Kim et al. (2016), Jo et al. (2016), and Samsung (2015). Kim et al. (2016) demonstrate between 5x and 47x performance improvement for scans and joins. Jo et al. (2016) describe a similar approach for In-Storage Computing based on the Samsung PM1725 SSD with ISC option (Samsung 2015) integrated in MariaDB. Other approaches (Cho et al. 2013b; Tiwari et al. 2013). Tiwari et al. (2013) stress the importance of in situ processing.

Woods et al. (2014, 2013) demonstrate with Ibex an intelligent storage engine for commodity-relational databases. By off-loading complex queries operators, they tackle the bandwidth bottlenecks arising when moving large amounts of data from storage to processing nodes. In addition, the energy consumption is reduced due to the usage of FPGAs rather than general-purpose processors. Ibex supports aggregation (GROUP BY), projection, and selection. Najafi et al. (2013) and Sadoghi et al. (2012) explore approaches for flexible query processing on FPGAs.

JAFAR is an *Active Storage* approach for column stores by Xi et al. (2015) and Babarinsa and Idreos (2015). JAFAR is based on MonetDB and aims at reducing data transfers, hence pushing size reducing DB operators such as selections; joins are not considered. Xi et al. (2015) stress the importance of on-chip accelerators but do not consider *Active Storage* and accelerators for complex computations in situ.

Memory: Processing-In-Memory (PIM)

Manufacturing costs of DRAM decrease, and memory volume increases steadily, while access latencies improve at a significantly lower rate yielding the so-called Memory Wall (Wulf and McKee 1995). On the one hand, technologies such as Hybrid Memory Cube (HMC) attempt to address this issue by locating processing units

close to memory and by utilizing novel interfaces alike. On the other hand, new types of memory are introduced, characterized by an even higher density and therefore larger volumes, shorter latencies, and higher bandwidth and internal parallelism, as well as non-volatile persistence behavior.

Balasubramonian (2016) discusses in his article the features that can be meaningfully added to memory devices. Not only do these features execute parts of an application, but they may also take care of auxiliary operations that maintain high efficiency, reliability, and security. Research combining memory technologies with the Active Storage concept in general, often referred to as Processing-In-Memory (PIM), is very versatile. In the late 1990 (Patterson et al. 1997), proposed IRAM as a first attempt to address the Memory Wall, by unifying processing logic and DRAM, starting with general research question of the computer science like communication, interfaces, cache coherence, or address schemes. Hall et al. (1999) purpose combining their Data-IntensiVe Architecture (DIVA) PIM memories with external host processors and defining a PIM-to-PIM interconnect; Vermij et al. (2017) present an extension to the CPU architecture to enable NDP capabilities close to the main memory by introducing a new component attached to the system bus responsible for the communication; Boroumand et al. (2017) propose a new hardware cache coherence mechanism designed specifically for PIM; Picorel et al. (2017) show that the historically important flexibility to map any virtual page to any page frame is unnecessary regarding NDP and introduce Distributed Inverted Page Table (DIPTA) as an alternative near-memory structure.

Studying the upcomming new interface HMC, Azarkhish et al. (2017) analyzed its support for NDP in a modular and flexible fashion. The authors propose a fully backward compatible extension to the standard HMC called smart memory cube and design a high-bandwidth, low-latency, and AXI(4)-compatible logic base interconnect, featuring a novel address scrambling mechanism for the reduction in

vault/bank conflicts. A completely different approach to tackle Active Storage in today's memory is presented in Gao et al. (2016b). It introduces DRAF, an architecture for bit-level reconfigurable logic that uses DRAM subarrays to implement dense lookup tables because FPGAs introduce significant area and power overheads, making it difficult to use them in datacenter servers. Leaving the existing sequential programming models in touch by extending the instruction set architecture, Ahn et al. (2015) proposes new PIM-enabled instructions. Firstly, the proposed instruction set is interoperable with existing programming models, cache coherence protocols, and virtual memory mechanisms. Secondly, the instructions can be executed either in-memory or on the processors depending on the data locality. A conceptual near-memory acceleration architecture is presented by Kim et al. (2017b) claiming the need for adopting a high-level synthesis approach. In Lim and Park (2017), kernel operations that can greatly improve with PIM are analyzed resulting in the necessity of three categories of processing engines for NDP logic – in-order core, a coarse-grain reconfigurable processor (CGRA), and dedicated hardware.

Proposing Caribou, an intelligent distributed storage layer, István et al. (2017) target NDP on DRAM/NVRAM storage over the network through a simple key-value store interface. Utilizing FPGAs, each storage node provides high-bandwidth NDP capabilities and fault tolerance through replication by Zookeeper's atomic broadcasts.

The application of the Active Storage concept on memories besides data management is often based on analytical scenarios or neural networks but comprises a variety of different approaches. Gao et al. (2016a) develop hardware and software for an NDP architecture for in-memory analytic frameworks, including MapReduce, graph processing, and deep neural networks. One year later, Gao et al. (2017) presents the hardware architecture and software scheduling and partitioning techniques for TETRIS, a scalable neural network accelerator using 3D memory. For a similar use case, (Chi et al. 2016) proposes

PRIME, providing microarchitecture and circuit designs for a ReRAM-based PIM architecture enabling morphable functions with insignificant area overhead. A compiler-based allocation strategy approach for PIM architectures is proposed by Memoultion Wang et al. (2017). Focusing on convolutional neural networks, it offers thread-level parallelism that can fully exploit the computational power-embedded processors. Another hardware/software co-design for data analytics is presented by the Mondrian Data Engine (Drummond et al. 2017). It focuses on sequential access patterns to enable simple hardware that access memory in streams. A standardization of NDP architecture, in order for PIM stacks to be used for different GPU architectures is proposed by Kim et al. (2017a). Their approach intend to allow data to be spread across multiple memory stacks as is the norm in high-performance systems.

Active Network

Having only the slight variation that data is not persisted at any time but rather streamed through, active networks are another very widespread application of the Active Storage concept. Powerful processing elements near the network adapters or often integrated to the network controller itself as a System-on-Chip (SoC) is not solely responsible for the conventional protocol interpretation anymore but also take over further tasks like security verifications and scheduling of in-transit services and data processing or simply improving the network performance.

Tennenhouse and Wetherall (1996) first introduced the term Active Network as an approach for performing sophisticated computation within the network. By injecting customized program features into the nodes of the network, it is possible to execute these at each traversed network router/switch. Continuing the research of Tennenhouse and Wetherall (1996) and Sykora and Koutny (2010) present an Active Network node called Smart Active Node (SAN). Thereby, they focus on its ability to translate data flow transparently between IP network and active network to further improve performance of IP applications.

Often Active Network is also referred as software-defined network (SDN) and comprises

already an advanced state of research. Especially the area around security comprises progressive research in authentication and authorization, access control, threats, and DoS attacks as summarized by Ahmad et al. (2015). But also the utilization of RDMA-capable network became a trend since the demand on higher bandwidth arose with the introduction of dedicated GPUs in the computation. Ren et al. (2017) propose iRDMA, an RDMA-based parameter server architecture optimized for high-performance network environment supporting both GPU- and CPU-based training.

Cross-References

- ▶ [Big Data and Exascale Computing](#)
- ▶ [Computer Architecture for Big Data](#)
- ▶ [Emerging Hardware Technologies](#)
- ▶ [Energy Implications of Big Data](#)

References

- Acharya A, Uysal M, Saltz J (1998) Active disks: Programming model, algorithms and evaluation. In: Proceedings of the eighth international conference on architectural support for programming languages and operating systems, ASPLOS VIII, pp 81–91
- Ahmad I, Namal S, Yliantila M, Gurtova A (2015) Security in software defined networks: a survey. *IEEE Commun Surv Tutorials* 17(4):2317–2346
- Ahn J, Yoo S, Mutlu O, Choi K (2015) PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In: Proceeding of 42nd annual international symposium on computer architecture (ISCA'15), pp 336–348
- Azarkish E, Pfister C, Rossi D, Loi I, Benini L (2017) Logic-base interconnect design for near memory computing in the smart memory cube. *IEEE Trans Very Large Scale Integr VLSI Syst* 25:210–223
- Babarinsa OO, Idreos S (2015) Jafar: near-data processing for databases. In: SIGMOD
- Balasubramonian R (2016) Making the case for feature-rich memory systems: the march toward specialized systems. *IEEE Solid-State Circuits Mag* 8(2):57–65
- Balasubramonian R, Chang J, Manning T, Moreno JH, Murphy R, Nair R, Swanson S (2014) Near-data processing: insights from a micro-46 workshop. *IEEE Micro* 34(4):36–42
- Boral H, DeWitt DJ (1983) Database machines: an idea whose time has passed? A critique of the future of database machines. In: Leilich H-O, Missikoff M (eds) Database machines. Springer, Berlin/Heidelberg, pp 166–187
- Boroumand A, Ghose S, Patel M, Hassan H, Lucia B, Hsieh K, Malladi KT, Zheng H, Mutlu O (2017) LazyPIM: an efficient cache coherence mechanism for processing-in-memory. *IEEE Comput Archit Lett* 16(1):46–50
- Chen C, Chen Y (2012) Dynamic active storage for high performance I/O. In: 2012 41st international conference on Parallel Processing. IEEE, pp 379–388
- Chi P, Li S, Xu C, Zhang T, Zhao J, Liu Y, Wang Y, Xie Y (2016) PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In: Proceeding of 2016 43rd international symposium on computer architecture (ISCA 2016), pp 27–39
- Cho BY, Jeong WS, Oh D, Ro WW (2013a) Xsd: accelerating mapreduce by harnessing the GPU inside an SSD. In: WoNDP: 1st workshop on near-data processing in conjunction with IEEE MICRO-46
- Cho S, Park C, Oh H, Kim S, Yi Y, Ganger GR (2013b) Active disk meets flash: a case for intelligent SSDs. In: Proceeding of ICS, pp 91–102
- DeWitt D, Gray J (1992) Parallel database systems: the future of high performance database systems. *Commun ACM* 35(6):85–98
- Do J, Kee YS, Patel JM, Park C, Park K, DeWitt DJ (2013) Query processing on smart SSDs: opportunities and challenges. In: Proceeding of SIGMOD, pp 1221–1230
- Drumond M, Daglis A, Mirzadeh N, Ustiugov D, Picorel J, Falsafi B, Grot B, Pnevmatikatos D (2017) The mondrian data engine. *ACM SIGARCH Comput Archit News* 45(2):639–651
- Fan S, He Z, Tan H (2016) An active storage system with dynamic task assignment policy. In: 2016 12th international conference on natural computation fuzzy system and knowledge discovery (ICNC-FSKD 2016), pp 1421–1427
- Gao M, Ayers G, Kozyrakis C (2016a) Practical near-data processing for in-memory analytics frameworks. Parallel architecture and compilation techniques – Conference proceedings, PACT 2016-March, pp 113–124
- Gao M, Delimitrou C, Niu D, Malladi KT, Zheng H, Brennan B, Kozyrakis C (2016b) DRAF: a low-power DRAM-based reconfigurable acceleration fabric. In: 2016 ACM/IEEE 43rd annual international symposium on computer architecture. IEEE, pp 506–518
- Gao M, Pu J, Yang X, Horowitz M, Kozyrakis C (2017) TETRIS: scalable and efficient neural network acceleration with 3D memory. *ASPLOS* 51(2):751–764
- Hall M, Kogge P, Koller J, Diniz P, Chame J, Draper J, LaCoss J, Granacki J, Brockman J, Srivastava A, Athas W, Freeh V, Shin J, Park J (1999) Mapping irregular applications to DIVA, a PIM-based data-intensive architecture. In: ACM/IEEE conference on supercomputing (SC 1999), p 57
- Hardavellas N, Ferdman M, Falsafi B, Ailamaki A (2011) Toward dark silicon in servers. *IEEE Micro* 31(4):6–15
- Hsieh K, Ebrahim E, Kim G, Chatterjee N, O'Connor M, Vijaykumar N, Mutlu O, Keckler SW (2016)

- Transparent offloading and mapping (TOM): enabling programmer-transparent near-data processing in GPU systems. In: Proceeding of 2016 43rd international symposium on computer architecture (ISCA 2016), pp 204–216
- István Z, Sidler D, Alonso G (2017) Caribou: intelligent distributed storage. *Proc VLDB Endow* 10(11): 1202–1213
- Jo I, Bae DH, Yoon AS, Kang JU, Cho S, Lee DDG, Jeong J (2016) Yoursql: a high-performance database system leveraging in-storage computing. *Proc VLDB Endow* 9:924–935
- Keeton K, Patterson DA, Hellerstein JM (1998) A case for intelligent disks (idisks). *SIGMOD Rec* 27(3):42–52
- Kim G, Chatterjee N, O'Connor M, Hsieh K (2017a) Toward standardized near-data processing with unrestricted data placement for GPUs. In: Proceeding of international conference on high performance computing networking, storage and analysis (SC'17), pp 1–12
- Kim NS, Chen D, Xiong J, Hwu WMW (2017b) Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era. *IEEE Micro* 37(4):10–18
- Kim S, Oh H, Park C, Cho S, Lee SW, Moon B (2016) In-storage processing of database scans and joins. *Inf Sci* 327(C):183–200
- Korinth J, Chevallerie Ddl, Koch A (2015) An open-source tool flow for the composition of reconfigurable hardware thread pool architectures. In: Proceedings of the 2015 IEEE 23rd annual international symposium on field-programmable custom computing machines (FCCM'15). IEEE Computer Society, Washington, DC, pp 195–198
- Kotra JB, Guttman D, Chidambaram Nachiappan N, Kan-demir MT, Das CR (2017) Quantifying the potential benefits of on-chip near-data computing in manycore processors. In: 2017 IEEE 25th international symposium on modeling, analysis, and simulation of computer and telecommunication system, pp 198–209
- Lim H, Park G (2017) Triple engine processor (TEP): a heterogeneous near-memory processor for diverse kernel operations. *ACM Ref ACM Trans Arch Code Optim Artic* 14(4):1–25
- Muramatsu B, Gierschi S, McMullan F, Weimar S, Klotz G (2004) If you build it, will they come? In: Proceeding of 2004 joint ACM/IEEE Conference on digital libraries (JCDL'04) p 396
- Najafi M, Sadoghi M, Jacobsen HA (2013) Flexible query processor on FPGAs. *Proc VLDB Endow* 6(12):1310–1313
- Patterson D, Anderson T, Cardwell N, Fromm R, Keeton K, Kozyrakis C, Thomas R, Yelick K (1997) A case for intelligent ram. *IEEE Micro* 17(2):34–44
- Petrov I, Almeida G, Buchmann A, Ulrich G (2010) Building large storage based on flash disks. In: Proceeding of ADMS'10
- Picorel J, Jevdjic D, Falsafi B (2017) Near-Memory Address Translation. In: 2017 26th international conference on Parallel architectures and compilation techniques, pp 303–317, 1612.00445
- Ren Y, Wu X, Zhang L, Wang Y, Zhang W, Wang Z, Hack M, Jiang S (2017) iRDMA: efficient use of RDMA in distributed deep learning systems. In: IEEE 19th international conference on high performance computing and communications, pp 231–238
- Riedel E, Gibson GA, Faloutsos C (1998) Active storage for large-scale data mining and multimedia. In: Proceedings of the 24rd international conference on very large data bases (VLDB'98), pp 62–73
- Sadoghi M, Javed R, Tarafdar N, Singh H, Palaniappan R, Jacobsen HA (2012) Multi-query stream processing on FPGAs. In: 2012 IEEE 28th international conference on data engineering, pp 1229–1232
- Samsung (2015) In-storage computing. http://www.flash-memorystandardsummit.com/English/Collaterals/Proceedings/2015/20150813_S301D_Ki.pdf
- Seshadri S, Gahagan M, Bhaskaran S, Bunker T, De A, Jin Y, Liu Y, Swanson S (2014) Willow: a user-programmable SSD. In: Proceeding of OSDI'14
- Sivathanu M, Bairavasundaram LN, Arpacı-Dusseau AC, Arpacı-Dusseau RH (2005) Database-aware semantically-smart storage. In: Proceedings of the 4th conference on USENIX conference on file and storage technologies (FAST'05), vol 4, pp 18–18
- Sykora J, Koutny T (2010) Enhancing performance of networking applications by IP tunneling through active networks. In: 9th international conference on networks (ICN 2010), pp 361–364
- Szalay A, Gray J (2006) 2020 computing: science in an exponential world. *Nature* 440:413–414
- Tennenhouse DL, Wetherall DJ (1996) Towards an active network architecture. *ACM SIGCOMM Comput Commun Rev* 26(2):5–17
- Tiwari D, Boboila S, Vazhkudai SS, Kim Y, Ma X, Desnoyers PJ, Solihin Y (2013) Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In: Proceeding of FAST, pp 119–132
- Vermij E, Fiorin L, Jongerius R, Hagleitner C, Lunteren JV, Bertels K (2017) An architecture for integrated near-data processors. *ACM Trans Archit Code Optim* 14(3):30:1–30:25
- Wang Y, Zhang M, Yang J (2017) Towards memory-efficient processing-in-memory architecture for convolutional neural networks. In: Proceeding 18th ACM SIGPLAN/SIGBED conference on languages compilers, and tools for embedded systems (LCTES 2017), pp 81–90
- Woods L, Teubner J, Alonso G (2013) Less watts, more performance: an intelligent storage engine for data appliances. In: Proceeding of SIGMOD, pp 1073–1076
- Woods L, István Z, Alonso G (2014) Ibex: an intelligent storage engine with support for advanced sql offloading. *Proc VLDB Endow* 7(11):963–974
- Wulf WA, McKee SA (1995) Hitting the memory wall: implications of the obvious. *SIGARCH CAN* 23(1):20–24
- Xi SL, Babarinsa O, Athanassoulis M, Idreos S (2015) Beyond the wall: near-data processing for databases. In: Proceeding of DaMoN, pp 2:1–2:10

Ad Hoc Benchmark

- ▶ [TPC-H](#)

Ad Hoc Query Processing

- ▶ [Robust Data Partitioning](#)

Adaptive Partitioning

- ▶ [Robust Data Partitioning](#)

Adaptive Windowing

Ricard Gavaldà

Universitat Politècnica de Catalunya, Barcelona,
Spain

Synonyms

[ADWIN algorithm](#)

Definitions

Adaptive Windowing is a technique used for the online analysis of data streams to manage changes in the distribution of the data. It uses the standard idea of sliding window over the data, but, unlike other approaches, the size of the window is not fixed and set a priori but changed dynamically as a function of the data. The window is maintained at all times to the maximum length consistent with the assumption that there is no change in the data contained in it.

Context

Many modern sources of data are best viewed as data streams: a potentially infinite sequence

of data items that arrive one at a time, usually at high and uncontrollable speed. One wants to perform various analysis tasks on the stream in an online, rather than batch, fashion. Among these tasks, many consist of building models such as creating a predictor, forming clusters, or discovering frequent patterns. The source of data may evolve over time, that is, its statistical properties may vary, and often one is interested in keeping the model accurate with respect to the current distribution of the data, rather than that of the past data.

The ability of model-building methods to handle evolving data streams is one of the distinctive concerns of data stream mining, compared to batch data mining (Gama et al. 2014; Ditzler et al. 2015; Bifet et al. 2018). Most of the approaches to this problem in the literature fall into one of the following three patterns or a combination thereof.

One, the algorithm keeps a sliding window over the stream that stores a certain quantity of the most recently seen items. The algorithm then is in charge of keeping the model accurate with respect to the items in the window. *Sliding* means that every newly arrived item is added to the front of the window and that the oldest elements are dropped from the tail of the window. Dropping policies may vary. To keep a window of a constant size (denoted W hereafter), one stores the first W elements and then drops exactly one element for each one that is added.

Two, each item seen so far is associated with a weight that changes over time. The model-building algorithm takes into account the weight of each element in maintaining its model, so that elements with higher weight influence more the model behavior. One can, for example, fix a constant $\lambda < 1$ called the *decay factor* and establish that the importance of every item gets decreased (multiplied) by λ at each time step; this implies that the importance of the item that arrived t time steps ago is λ^t its initial one, that is, weights decrease exponentially fast. This policy is the basis of the EWMA (Exponentially Weighted Moving Average) estimator for the average of some statistic of the stream.

Three, the model builder monitors the stream with a *change detection* algorithm that raises a

flag when it finds evidence that the distribution of the stream items has changed. When this happens, the model builder revises the current model or discards the model and builds a new one with fresh data. Usually, change detection algorithms monitor one statistic or a small number of statistics of the stream, so they will not detect every possible kind of change, which is in general computationally unfeasible. Two easy-to-implement change detection algorithms for streams of real values are the CUSUM (Cumulative Sum) and Page-Hinkley methods (Basseville and Nikiforov 1993; Gama et al. 2014). Roughly speaking, both methods monitor the average of the items in the stream; when the recent average differs from the historical average by some threshold related to the standard deviation, they declare change.

Methods such as EWMA, CUSUM, and Page-Hinkley store a constant amount of real values, while window-based methods require, if implemented naively, memory linear in W . On the other hand, keeping a window provides more information usable by the model builder, namely, the instances themselves.

A disadvantage of all three methods as described is that they require the user to provide parameters containing assumptions about the magnitude or frequency of changes. Fixing a window size to have size W means that the user expects the last W items to be relevant, so that there is little change within them, but that items older than W are suspect of being irrelevant. Fixing a parameter λ in the EWMA estimator to a value close to 1 indicates that change is expected to be rare or slow, while a value closer to 0 suggests that change may be frequent or abrupt. A similar assumption can be found in the choice of parameters for the CUSUM and Page-Hinkley tests.

In general, these methods face the trade-off between reaction time and variance in the data: The user would like them to react quickly to changes (which happens with, e.g., smaller values of W and λ) but also have a low number of false positives when no change occurs (which is achieved with larger values of W and λ). In the case of sliding windows, in general one wants to have larger values of W when no change occurs, because models built from more data tend to be

more accurate, but smaller values of W when the data is changing, so that the model ignores obsolete data. These trade-offs are investigated by Kuncheva and Žliobaitė (2009).

Methods

Adaptive windowing schemes use sliding windows whose size increases or decreases in response to the change observed in the data. They intend to free the user from having to guess expected rates of change in the data, which may lead to poor performance if the guess is incorrect or if the rate of change is different at different moments. Three methods are reviewed in this section, particularly the ADWIN method.

The Drift Detection Method (DDM) proposed by Gama et al. (2004) applies to the construction of two-class predictive models. It is based on the theoretical and practical observation that the empirical error of a predictive model should decrease or remain stable as the model is built with more and more data from a stationary distribution, assuming one controls overfitting. Therefore, when the empirical error instead increases, this is evidence that the distribution in the data has changed.

More precisely, let p_t denote the error rate of the predictor at time t , and $s_t = \sqrt{p_t(1-p_t)/t}$ its standard deviation. DDM stores the smallest value p_{\min} of the error rates observed up to time t , and the standard deviation s_{\min} at that point. Then at time t :

- If $p_t + s_t \geq p_{\min} + 2 \cdot s_{\min}$, DDM declares a warning. It starts storing examples in anticipation of a possible declaration of change.
- If $p_t + s_t \geq p_{\min} + 3 \cdot s_{\min}$, DDM declares a change. The current predictor is discarded and a new one is built using the stored examples. The values for p_{\min} and s_{\min} are reset as well.

This approach is generic and fast enough for the use in the streaming setting, but it has the drawback that it may be too slow in responding to changes. Indeed, since p_t is computed on the

basis of all examples since the last change, it may take many observations after the change to make p_t significantly larger than p_{\min} . Also, for slow change, the number of examples retained in memory may become large.

An evolution of this method that uses EWMA to estimate the errors is presented and thoroughly analyzed by Ross et al. (2012).

The OLIN method due to Last (2002) and Cohen et al. (2008) also adjusts dynamically the size of the sliding window used to update a predictive model, in order to adapt it to the rate of change in nonstationary data streams. OLIN uses the statistical significance of the difference between the training and the validation accuracy of the current model as an indicator of data stability. Higher stability means that the window can be enlarged to use more data to build a predictor, and lower stability implies shrinking the window to discard stale data. Although described for one specific type of predictor (“Information Networks”) in Last (2002) and Cohen et al. (2008), the technique should apply many other types.

The ADWIN (ADaptive WINdowing) algorithm is due to Bifet and Gavaldà (2007) and Bifet (2010). Its purpose is to be a self-contained module that can be used in the design of data stream algorithms (for prediction or classification, but also for other tasks) to detect and manage change in a well-specified way. In particular, it wants to resolve the trade-off between fast reaction to change and reduced false alarms without relying on the user guessing an ad hoc parameter. Intuitively, the ADWIN algorithm resolves this trade-off by checking change at many scales simultaneously or trying many sliding window sizes simultaneously. It should be used when the scale of change rate is unknown, and this is problematic enough to compensate a moderate increase in computational effort.

More precisely, ADWIN maintains a sliding window of real numbers that are derived from the data stream. For example, elements in the window could be W bits indicating whether the current predictive model was correct on the last W stream items; the window then can be used to estimate the current error rate of the predictor. In a clustering task, it could instead keep track of the

fraction of outliers or cluster quality measures, and in a frequent pattern mining task, the number of frequent patterns that appear in the window. Significant variation inside the window of any of these measures indicates distribution change in the stream. ADWIN is parameterized by a confidence parameter $\delta \in (0, 1)$ and a statistical test $T(W_0, W_1, \delta)$; here W_0 and W_1 are two windows, and T decides whether they are likely to come from the same distribution. A good test should satisfy the following criteria:

- If W_0 and W_1 were generated from the same distribution (no change), then with probability at least $1 - \delta$ the test says “no change.”
- If W_0 and W_1 were generated from two different distributions whose average differs by more than some quantity $\epsilon(W_0, W_1, \delta)$, then with probability at least $1 - \delta$ the test says “change.”

When there has been change in the average but its magnitude is less than $\epsilon(W_0, W_1, \delta)$, no claims can be made on the validity of the test’s answer. Observe that in reasonable tests, ϵ decreases as the sizes of W_0 and W_1 increase, that is, as the test sees larger samples. ADWIN applies the test to a number of partitions of its sliding window into two parts, W_0 containing the oldest elements and W_1 containing the newer ones. Whenever $T(W_0, W_1, \delta)$ returns “change”, ADWIN drops W_0 , so the sliding window becomes W_1 . In this way, at all times, the window is kept of the maximum length such that there is no proof of change within it. In times without change, the window can keep growing indefinitely (up to a maximum size, if desired).

In order to be efficient in time and memory, ADWIN represents its sliding window in a compact way, using the Exponential Histogram data structure due to Datar et al. (2002). This structure maintains a summary of a window by means of a chain of buckets. Older bits are summarized and compressed in coarser buckets with less resolution. A window of length W is stored in only $O(k \log W)$ buckets, each using a constant amount of memory words, and yet

the histogram returns an approximation of the average of the window values that is correct up to a factor of $1/k$. ADWIN does not check all partitions of its window into pairs (W_0, W_1), but only those at bucket boundaries. Therefore, it performs $O(k \log W)$ tests on the sliding window for each stream item. The standard implementation of ADWIN uses $k = 5$ and may add the rule that checks are only performed only every t number of items for efficiency – at the price of a delay of up to t time steps in detecting a change.

In Bifet and Gavaldà (2007) and Bifet (2010), a test based on the so-called Hoeffding bound is proposed, which can be rigorously proved to satisfy the conditions above for a “good test.” Based on this, rigorous guarantees on the false positive rate and false negative rate of ADWIN are proved in Bifet and Gavaldà (2007) and Bifet (2010). However, this test is quite conservative and will be slow to detect change. In practice, tests based on the normal approximation of a binomial distribution should be used, obtaining faster reaction time for a desired false positive rate.

Algorithms for mining data streams will probably store their own sliding window of examples to revise/rebuild their models. One or several instances of ADWIN can be used to inform the algorithm of the occurrence of change and the optimal window size it should use. The time and memory overhead is moderate (logarithmic in the size of the window) and often negligible compared with the cost of the main algorithm itself.

Several change detection methods for streams were evaluated by Gama et al. (2009). The conclusions were that Page-Hinkley and ADWIN were the most appropriate. Page-Hinkley exhibited a high rate of false alarms, and ADWIN used more resources, as expected.

Examples of Application

ADWIN has been applied in the design of streaming algorithms and in applications that need to deal with nonstationary streams.

At the level of algorithm design, it was used by Bifet and Gavaldà (2009) to give a more adaptive version of the well-known CVFDT algorithm for building decision trees from streams due to Hulten et al. (2001); ADWIN improves it by replacing hard-coded constants in CVFDT for the sizes of sliding windows and the duration of testing and training phases with data-adaptive conditions. A similar approach was used by Bifet et al. (2010b) for regression trees using perceptrons at the leaves. In Bifet et al. (2009a,b, 2010a, 2012), ADWIN was used in the context of ensemble classifiers to detect when a member of the ensemble is underperforming and needs to be replaced. In the context of pattern mining, ADWIN was used to detect change and maintain the appropriate sliding window size in algorithms that extract frequent graphs and frequent trees from streams of graphs and XML trees (Bifet et al. 2011b; Bifet and Gavaldà 2011). In the context of process mining, ADWIN is used by Carmona and Gavaldà (2012) to propose a mechanism that helps in detecting changes in the process, localize and characterize the change once it has occurred, and unravel process evolution.

ADWIN is a very generic, domain-independent mechanism that can be plugged into a large variety of applications. Some examples include the following:

- Bakker et al. (2011) in an application to detect stress situations in the data from wearable sensors
- Bifet et al. (2011a) in application to detect sentiment change in Twitter streaming data
- Pechenizkiy et al. (2009) as the basic detection mechanism in a system to control the stability and efficiency of industrial fuel boilers
- Talavera et al. (2015) in an application to segmentation of video streams

Future Research

A main research problem continues to be the efficient detection of change in multidimensional data. Algorithms as described above (OLIN,

DDM, and ADWIN) deal, strictly speaking, with the detection of change in a unidimensional stream of real values; it is assumed that this stream of real values, derived from the real stream, will change significantly when there is significant change in the multidimensional stream.

Several instances of these detectors can be created to monitor different parts of the data space or to monitor different summaries or projections thereof, as in, e.g., Carmona and Gavaldà (2012). However, there is no guarantee that all real changes will be detected in this way. Papapetrou et al. (2015) and Muthukrishnan et al. (2007) among others have proposed efficient schemes to directly monitor change in change in multidimensional data. However, the problem in its full generality is difficult to scale to high dimensions and arbitrary change, and research in more efficient mechanisms usable in streaming scenarios is highly desirable.

Cross-References

- ▶ [Definition of Data Streams](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Management of Time](#)

References

- Bakker J, Pechenizkiy M, Sidorova N (2011) What's your current stress level? Detection of stress patterns from GSR sensor data. In: 2011 IEEE 11th international conference on data mining workshops (ICDMW), Vancouver, 11 Dec 2011, pp 573–580. <https://doi.org/10.1109/ICDMW.2011.178>
- Basseville M, Nikiforov IV (1993) Detection of abrupt changes: theory and application. Prentice-Hall, Upper Saddle River. <http://people.irisa.fr/Michele.Basseville/kniga/>. Accessed 21 May 2017
- Bifet A (2010) Adaptive stream mining: pattern learning and mining from evolving data streams, frontiers in artificial intelligence and applications, vol 207. IOS Press. <http://www.booksonline.iospress.nl/Content/View.aspx?piid=14470>
- Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the seventh SIAM international conference on data mining, 26–28 Apr 2007, Minneapolis, pp 443–448. <https://doi.org/10.1137/1.9781611972771.42>
- Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: Advances in intelligent data analysis VIII, proceedings of the 8th international symposium on intelligent data analysis, IDA 2009, Lyon, Aug 31–Sept 2 2009, pp 249–260. https://doi.org/10.1007/978-3-642-03915-7_22
- Bifet A, Gavaldà R (2011) Mining frequent closed trees in evolving data streams. *Intell Data Anal* 15(1):29–48. <https://doi.org/10.3233/IDA-2010-0454>
- Bifet A, Holmes G, Pfahringer B, Gavaldà R (2009a) Improving adaptive bagging methods for evolving data streams. In: Advances in machine learning, proceedings of the first Asian conference on machine learning, ACML 2009, Nanjing, 2–4 Nov 2009, pp 23–37. https://doi.org/10.1007/978-3-642-05224-8_4
- Bifet A, Holmes G, Pfahringer B, Kirkby R, Gavaldà R (2009b) New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '09. ACM, New York, pp 139–148. <http://doi.acm.org/10.1145/1557019.1557041>
- Bifet A, Holmes G, Pfahringer B (2010a) Leveraging bagging for evolving data streams. In: European conference on machine learning and knowledge discovery in databases, proceedings, part I of the ECML PKDD 2010, Barcelona, 20–24 Sept 2010, pp 135–150. https://doi.org/10.1007/978-3-642-15880-3_15
- Bifet A, Holmes G, Pfahringer B, Frank E (2010b) Fast perceptron decision tree learning from evolving data streams. In: Advances in knowledge discovery and data mining, proceedings, part II of the 14th Pacific-Asia conference, PAKDD 2010, Hyderabad, 21–24 June 2010, pp 299–310. https://doi.org/10.1007/978-3-642-13672-6_30
- Bifet A, Holmes G, Pfahringer B, Gavaldà R (2011a) Detecting sentiment change in twitter streaming data. In: Proceedings of the second workshop on applications of pattern analysis, WAPA 2011, Castro Urdiales, 19–21 Oct 2011, pp 5–11. <http://jmlr.csail.mit.edu/proceedings/papers/v17/bifet11a/bifet11a.pdf>
- Bifet A, Holmes G, Pfahringer B, Gavaldà R (2011b) Mining frequent closed graphs on evolving data streams. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '11. ACM, New York, pp 591–599. <http://doi.acm.org/10.1145/2020408.2020501>
- Bifet A, Frank E, Holmes G, Pfahringer B (2012) Ensembles of restricted hoeffding trees. *ACM TIST* 3(2):30:1–30:20. <http://doi.acm.org/10.1145/2089094.2089106>
- Bifet A, Gavaldà R, Holmes G, Pfahringer B (2018) Machine learning for data streams, with practical examples in MOA. MIT Press, Cambridge. <https://mitpress.mit.edu/books/machine-learning-data-streams>
- Carmona J, Gavaldà R (2012) Online techniques for dealing with concept drift in process mining. In: Advances in intelligent data analysis XI – proceedings of the 11th international symposium, IDA 2012, Helsinki, 25–27 Oct 2012, pp 90–102. https://doi.org/10.1007/978-3-642-34156-4_10

- Cohen L, Avrahami-Bakish G, Last M, Kandel A, Kipersztok O (2008) Real-time data mining of non-stationary data streams from sensor networks. *Info Fusion* 9(3):344–353. <https://doi.org/10.1016/j.inffus.2005.05.005>
- Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. *SIAM J Comput* 31(6):1794–1813. <https://doi.org/10.1137/S0097539701398363>
- Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. *IEEE Comp Int Mag* 10(4):12–25. <https://doi.org/10.1109/MCI.2015.2471196>
- Gama J, Medas P, Castillo G, Rodrigues PP (2004) Learning with drift detection. In: Advances in artificial intelligence – SBIA 2004, proceedings of the 17th Brazilian symposium on artificial intelligence, São Luis, Sept 29–Oct 1 2004, pp 286–295. https://doi.org/10.1007/978-3-540-28645-5_29
- Gama J, Sebastião R, Rodrigues PP (2009) Issues in evaluation of stream learning algorithms. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 09), Paris, June 28–July 1 2009, pp 329–338. <http://doi.acm.org/10.1145/1557019.1557060>
- Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv* 46(4):44:1–44:37. <http://doi.acm.org/10.1145/2523813>
- Hulten G, Spencer L, Domingos PM (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining (KDD 01), San Francisco, 26–29 Aug 2001, pp 97–106. <http://portal.acm.org/citation.cfm?id=502512.502529>
- Kuncheva LI, Žliobaitė I (2009) On the window size for classification in changing environments. *Intell Data Anal* 13(6):861–872. <https://doi.org/10.3233/IDA-2009-0397>
- Last M (2002) Online classification of nonstationary data streams. *Intell Data Anal* 6(2):129–147. <http://content.iospress.com/articles/intelligent-data-analysis/ida00083>
- Muthukrishnan S, van den Berg E, Wu Y (2007) Sequential change detection on data streams. In: Workshops proceedings of the 7th IEEE international conference on data mining (ICDM 2007), 28–31 Oct 2007, Omaha, pp 551–550. <https://doi.org/10.1109/ICDMW.2007.89>
- Papapetrou O, Garofalakis MN, Deligiannakis A (2015) Sketching distributed sliding-window data streams. *VLDB J* 24(3):345–368. <https://doi.org/10.1007/s00778-015-0380-7>
- Pechenizkiy M, Bakker J, Žliobaitė I, Ivannikov A, Kärkkäinen T (2009) Online mass flow prediction in CFB boilers with explicit detection of sudden concept drift. *SIGKDD Explor* 11(2):109–116. <http://doi.acm.org/10.1145/1809400.1809423>
- Ross GJ, Adams NM, Tasoulis DK, Hand DJ (2012) Exponentially weighted moving average charts for detecting concept drift. *Pattern Recogn Lett* 33(2): 191–198. <https://doi.org/10.1016/j.patrec.2011.08.019>, erratum in *Pattern Recogn Lett* 33(16):2261 (2012)
- Talavera E, Dimiccoli M, Bolaños M, Aghaei M, Radeva P (2015) R-clustering for egocentric video segmentation. In: Pattern recognition and image analysis – 7th Iberian conference, proceedings of the IbPRIA 2015, Santiago de Compostela, 17–19 June 2015, pp 327–336. https://doi.org/10.1007/978-3-319-19390-8_37

A

Advancements in YARN Resource Manager

Konstantinos Karanasos, Arun Suresh, and Chris Douglas
Microsoft, Washington, DC, USA

Synonyms

Cluster scheduling; Job scheduling; Resource management

Definitions

YARN is currently one of the most popular frameworks for scheduling jobs and managing resources in shared clusters. In this entry, we focus on the new features introduced in YARN since its initial version.

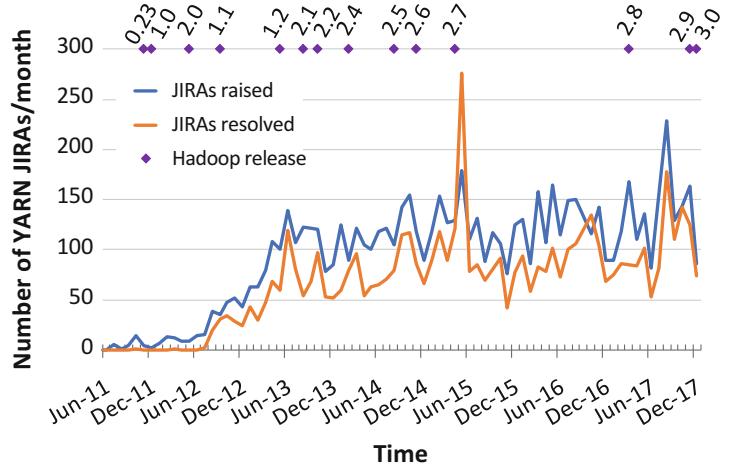
Overview

Apache Hadoop (2017), one of the most widely adopted implementations of MapReduce (Dean and Ghemawat 2004), revolutionized the way that companies perform analytics over vast amounts of data. It enables parallel data processing over clusters comprised of thousands of machines while alleviating the user from implementing complex communication patterns and fault tolerance mechanisms.

With its rise in popularity, came the realization that Hadoop's resource model for MapReduce, albeit flexible, is not suitable for every application, especially those relying on low-latency

Advancements in YARN Resource Manager, Fig. 1

Timeline of Apache Hadoop releases (on top) and number of raised and resolved tickets (JIRAs) per month on YARN



or iterative computations. This motivated decoupling the cluster resource management infrastructure from specific programming models and led to the birth of YARN (Vavilapalli et al. 2013). YARN manages cluster resources and exposes a generic interface for applications to request resources. This allows several applications, including MapReduce, to be deployed on a single cluster and share the same resource management layer.

YARN is a community-driven effort that was first introduced in Apache Hadoop in November 2011, as part of the 0.23 release. Since then, the interest of the community has continued unabated. Figure 1 shows that more than 100 tickets, i.e., JIRAs (YARN JIRA 2017), related to YARN are raised every month. A steady portion of these JIRAs are resolved, which shows the continuous community engagement. In the past year alone, 160 individuals have contributed code to YARN.

Moreover, YARN has been widely deployed across hundreds of companies for production purposes, including Yahoo! (Oath), Microsoft, Twitter, LinkedIn, Hortonworks, Cloudera, eBay, and Alibaba.

Since YARN's inception, we observe the following trends in modern clusters:

Application variety Users' interest has expanded from batch analytics applications (e.g., MapReduce) to include streaming, iterative

(e.g., machine learning), and interactive computations.

Large shared clusters Instead of using dedicated clusters for each application, diverse workloads are consolidated on clusters of thousands or even tens of thousands of machines. This consolidation avoids unnecessary data movement, allows for better resource utilization, and enables pipelines with different application classes.

High resource utilization Operating large clusters involves a significant cost of ownership. Hence, cluster operators rely on resource managers to achieve high cluster utilization and improve their return on investment.

Predictable execution Production jobs typically come with Service Level Objectives (SLOs), such as completion deadlines, which have to be met in order for the output of the jobs to be consumed by downstream services. Execution predictability is often more important than pure application performance when it comes to business-critical jobs.

This diverse set of requirements has introduced new challenges to the resource management layer. To address these new demands, YARN has evolved from a platform for batch analytics workloads to a production-ready, general-purpose resource manager that can support a wide range of applications and user requirements over large shared clusters. In the

remainder of this entry, we first give a brief overview of YARN’s architecture and dedicate the rest of the paper to the new functionality that was added to YARN these last years.

YARN Architecture

YARN follows a centralized architecture in which a single logical component, the resource manager (RM), allocates resources to jobs submitted to the cluster. The resource requests handled by the RM are intentionally generic, while specific scheduling logic required by each application is encapsulated in the application master (AM) that any framework can implement. This allows YARN to support a wide range of applications using the same RM component. YARN’s architecture is depicted in Fig. 2. Below we describe its main components. The new features, which appear in orange, are discussed in the following sections.

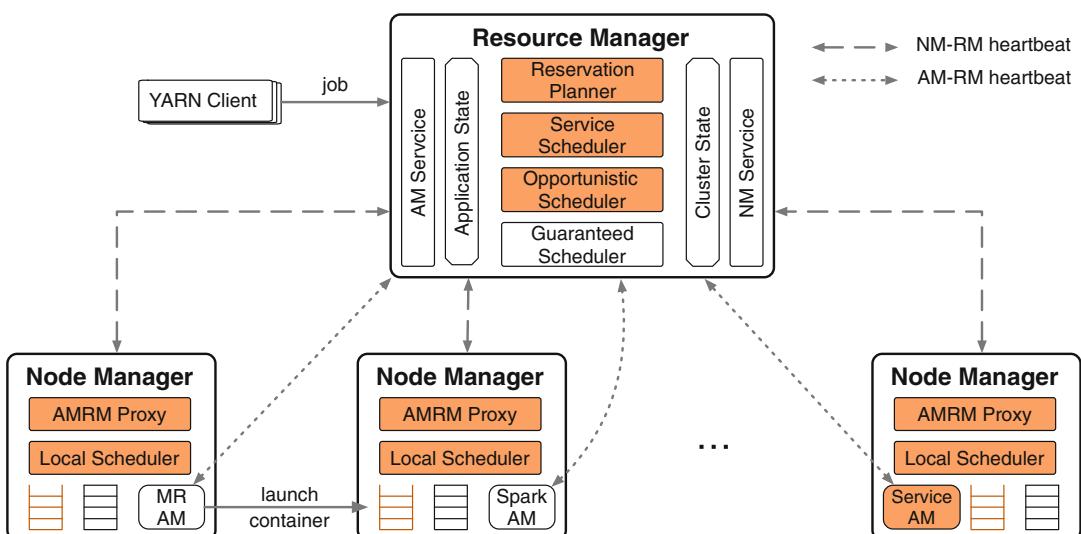
Node Manager (NM) The NM is a daemon running at each of the cluster’s worker nodes. NMs are responsible for monitoring resource availability at the host node, reporting faults,

and managing containers’ life cycle (e.g., start, monitor, pause, and kill containers).

Resource Manager (RM) The RM runs on a dedicated machine, arbitrating resources among various competing applications. Multiple RMs can be used for high availability, with one of them being the master. The NMs periodically inform the RM of their status, which is stored at the *cluster state*. The RM-NM communication is heartbeat-based for scalability. The RM also maintains the resource requests of all applications (*application state*). Given its global view of the cluster and based on application demand, resource availability, scheduling priorities, and sharing policies (e.g., fairness), the *scheduler* performs the matchmaking between application requests and machines and hands leases, called *containers*, to applications. A container is a logical resource bundle (e.g., 2 GB RAM, 1 CPU) bound to a specific node.

YARN includes two scheduler implementations, namely, the Fair and Capacity Schedulers. The former imposes fairness between applications, while the latter dedicates a share of the cluster resources to groups of users.

Jobs are submitted to the RM via the *YARN Client* protocol and go through an



Advancements in YARN Resource Manager, Fig. 2 YARN architecture and overview of new features (in orange)

admission control phase, during which security credentials are validated and various operational and administrative checks are performed.

Application Master (AM) The AM is the job orchestrator (one AM is instantiated per submitted job), managing all its life cycle aspects, including dynamically increasing and decreasing resource consumption, managing the execution flow (e.g., running reducers against the output of mappers), and handling faults. The AM can run arbitrary user code, written in any programming language. By delegating all these functions to AMs, YARN’s architecture achieves significant scalability, programming model flexibility, and improved upgrading/testing.

An AM will typically need to harness resources from multiple nodes to complete a job. To obtain containers, the AM issues resource requests to the RM via heartbeats, using the *AM Service* interface. When the scheduler assigns a resource to the AM, the RM generates a lease for that resource. The AM is then notified and presents the container lease to the NM for launching the container at that node. The NM checks the authenticity of the lease and then initiates the container execution.

In the following sections, we present the main advancements made in YARN, in particular with respect to resource utilization, scalability, support for services, and execution predictability.

Resource Utilization

In the initial versions of YARN, the RM would assign containers to a node only if there were unallocated resources on that node. This **guaranteed** type of allocation ensures that once an AM dispatches a container to a node, there will be sufficient resources for its execution to start immediately.

Despite the predictable access to resources that this design offers, it has the following shortcomings that can lead to suboptimal resource utilization:

Feedback delays The heartbeat-based AM-RM and NM-RM communications can cause idle node resources from the moment a container finishes its execution on a node to the moment an AM gets notified through the RM to launch a new container on that node.

Underutilized resources The RM assigns containers based on the allocated resources at each node, which might be significantly higher than the actually utilized ones (e.g., a 4GB container using only 2 GB of its memory).

In a typical YARN cluster, NM-RM heartbeat intervals are set to 3 s, while AM-RM intervals vary but are typically up to a few seconds. Therefore, feedback delays are more pronounced for workloads with short tasks.

Below we describe the new mechanisms that were introduced in YARN to improve cluster resource utilization. These ideas first appeared in the Mercury and Yaq systems (Karanasos et al. 2015; Rasley et al. 2016) and are part of Apache Hadoop as of version 2.9 (Opportunistic scheduling 2017; Distributed scheduling 2017).

Opportunistic containers Unlike guaranteed containers, opportunistic ones are dispatched to an NM, even if there are no available resources on that node. In such a case, the opportunistic containers will be placed in a newly introduced **NM queue** (see Fig. 2). When resources become available, an opportunistic container will be picked from the queue, and its execution will start immediately, avoiding any feedback delays.

These containers run with lower priority in YARN and will be preempted in case of resource contention for guaranteed containers to start their execution. Hence, opportunistic containers improve cluster resource utilization without impacting the execution of guaranteed containers. Moreover, whereas the original NM passively executes conflict-free commands from the RM, a modern NM uses these two-level priorities as inputs to local scheduling decisions. For instance, low-priority jobs with non-strict execution guarantees or tasks off the critical path of a DAG, are good candidates for opportunistic containers.

The AMs currently determine the execution type for each container, but the system could use automated policies instead. The AM can also request promotion of opportunistic containers to guarantee to protect them from preemption.

Hybrid scheduling Opportunistic containers can be allocated centrally by the RM or in a distributed fashion through a local scheduler that runs at each NM and leases containers on other NMs without contacting the RM. Centralized allocation allows for higher-quality placement decisions and sharing policies. Distributed allocation offers lower allocation latencies, which can be beneficial for short-lived containers. To prevent conflicts, guaranteed containers are always assigned by the RM.

To determine the least-loaded nodes for placing opportunistic containers, the RM periodically gathers information about the running and queued containers at each node and propagates this information to the local schedulers too. To account for occasional load imbalance across nodes, YARN performs dynamic rebalancing of queued containers.

Resource overcommitment Currently, opportunistic containers can be employed to avoid feedback delays. Ongoing development also focuses on overcommitting resources using opportunistic containers (Utilization-based scheduling 2017). In this scenario, opportunistic containers facilitate reclaiming overcommitted resources on demand, without affecting the performance and predictability of jobs that opt out of overcommitted resources.

Cluster Scalability

A single YARN RM can manage a few thousands of nodes. However, production analytics clusters at big cloud companies are often comprised of tens of thousands of machines, crossing YARN’s limits (Burd et al. 2017).

YARN’s scalability is constrained by the resource manager, as load increases proportionally to the number of cluster nodes and the appli-

cation demands (e.g., active containers, resource requests per second). Increasing the heartbeat intervals could improve scalability in terms of number of nodes, but would be detrimental to utilization (Vavilapalli et al. 2013) and would still pose problems as the number of applications increases.

Instead, as of Apache Hadoop 2.9 (YARN Federation 2017), a **federation-based** approach scales a single YARN cluster to tens of thousands of nodes. This approach divides the cluster into smaller units, called **subclusters**, each with its own YARN RM and NMs. The federation system negotiates with subcluster RMs to give applications the experience of a single large cluster, allowing applications to schedule their tasks to any node of the federated cluster.

The state of the federated cluster is coordinated through the **State Store**, a central component that holds information about (1) subcluster liveliness and resource availability via heartbeats sent by each subcluster RM, (2) the YARN subcluster at which each AM is being deployed, and (3) policies used to impose global cluster invariants and perform load rebalancing.

To allow jobs to seamlessly span subclusters, the federated cluster relies on the following components:

Router A federated YARN cluster is equipped with a set of routers, which hide the presence of multiple RMs from applications. Each application gets submitted to a router, which, based on a policy, determines the subcluster for the AM to be executed, gets the subcluster URL from the State Store, and redirects the application submission request to the appropriate subcluster RM.

AMRM Proxy This component runs as a service at each NM of the cluster and acts as a proxy for every AM-RM communication. Instead of directly contacting the RM, applications are forced by the system to access their local AMRM Proxy. By dynamically routing the AM-RM messages, the AMRM Proxy provides the applications with transparent access to multiple YARN RMs. Note that the AMRM Proxy is also used to implement the

local scheduler for opportunistic containers and could be used to protect the system against misbehaving AMs.

This federated design is scalable, as the number of nodes each RM is responsible for is bounded. Moreover, through appropriate policies, the majority of applications will be executed within a single subcluster; thus the number of applications that are present at each RM is also bounded. As the coordination between subclusters is minimal, the cluster's size can be scaled almost linearly by adding more subclusters. This architecture can provide tight enforcement of scheduling invariants within a subcluster, while continuous rebalancing across subclusters enforces invariants in the whole cluster.

A similar federated design has been followed to scale the underlying store (HDFS Federation 2017).

Long-Running Services

As already discussed, YARN's target applications were originally batch analytics jobs, such as MapReduce. However, a significant share of today's clusters is dedicated to workloads that include stream processing, iterative computations, data-intensive interactive jobs, and latency-sensitive online applications. Unlike batch jobs, these applications benefit from long-lived containers (from hours to months) to amortize container initialization costs, reduce scheduling load, or maintain state across computations. Here we use the term *services* for all such applications.

Given their long-running nature, these applications have additional demands, such as support for restart, in-place upgrade, monitoring, and discovery of their components. To avoid using YARN's low-level API for enabling such operations, users have so far resorted to AM libraries such as Slider (Apache Slider 2017). Unfortunately, these external libraries only partially solve the problem, e.g., due to lack of common standards for YARN to optimize resource demands

across libraries or version incompatibilities between the libraries and YARN.

To this end, the upcoming Apache Hadoop 3.1 release adds first-class support for long-running services in YARN, allowing for both traditional process-based and Docker-based containers. This service framework allows users to deploy existing services on YARN, simply by providing a JSON file with their service specifications, without having to translate those requirements into low-level resource requests at runtime.

The main component of YARN's service framework is the **container orchestrator**, which facilitates service deployment. It is an AM that, based on the service specification, configures the required requests for the RM and launches the corresponding containers. It deals with various service operations, such as starting components given specified dependencies, monitoring their health and restarting failed ones, scaling up and down component resources, upgrading components, and aggregating logs.

A **RESTful API server** is developed to allow users to manage the life cycle of services on YARN via simple commands, using framework-independent APIs. Moreover, a **DNS server** enables service discovery via standard DNS lookups and greatly simplifies service failovers.

Scheduling services Apart from the aforementioned support for service deployment and management, service owners also demand precise control of container placement to optimize the performance and resilience of their applications. For instance, containers of services are often required to be collocated (affinity) to reduce network costs or separated (anti-affinity) to minimize resource interference and correlated failures. For optimal service performance, even more powerful constraints are useful, such as complex intra- and inter-application constraints that collocate services with one another or put limits in the number of specific containers per node or rack.

When placing containers of services, cluster operators have their own, potentially conflicting, global optimization objectives. Examples include minimizing the violation of placement constraints, the resource fragmentation, the load

imbalance, or the number of machines used. Due to their long lifetimes, services can tolerate longer scheduling latencies than batch jobs, but their placement should not impact the scheduling latencies of the latter.

To enable high-quality placement of services in YARN, Apache Hadoop 3.1 introduces support for rich placement constraints (Placement constraints 2017).

Jobs with SLOs

In production analytics clusters, the majority of cluster resources is usually consumed by **production jobs**. These jobs must meet strict Service Level Objectives (SLOs), such as completion deadlines, for their results to be consumed by downstream services. At the same time, a large number of smaller **best-effort jobs** are submitted to the same clusters in an ad hoc manner for exploratory purposes. These jobs lack SLOs, but they are sensitive to completion latencies.

Resource managers typically allocate resources to jobs based on *instantaneous* enforcement of job priorities and sharing invariants. Although simpler to implement and impose, this instantaneous resource provisioning makes it challenging to meet job SLOs without sacrificing low latency for best-effort jobs.

To ensure that important production jobs will have predictable access to resources, YARN was extended with the notion of *reservations*, which provide users with the ability to reserve resources over (and ahead of) time. The ideas around reservations first appeared in Rayon (Curino et al. 2014) and are part of YARN as of Apache Hadoop 2.6.

Reservations This is a construct that determines the resource needs and temporal requirements of a job and translates the job's completion deadline into an SLO over predictable resource allocations. This is done ahead of the job's execution, aimed at ensuring a predictable and timely execution. To this end, YARN introduced a reservation definition language (RDL) to express a rich class of time-aware resource requirements, including

deadlines, malleable and gang parallelism, and inter-job dependencies.

Reservation planning and scheduling RDL provides a uniform and abstract representation of jobs' needs. Reservation requests are received ahead of a job's submission by the **reservation planner**, which performs online admission control. It accepts all jobs that can fit in the cluster agenda over time and rejects those that cannot be satisfied. Once a reservation is accepted by the planner, the scheduler is used to dynamically assign cluster resources to the corresponding job.

Periodic reservations Given that a high percentage of production jobs are recurring (e.g., hourly, daily, or monthly), YARN allows users to define periodic reservations, starting with Apache Hadoop 2.9. A key property of recurring reservations is that once a periodic job is admitted, each of its instantiations will have a predictable resource allocation. This isolates periodic production jobs from the noisiness of sharing.

Toward predictable execution The idea of recurring reservations was first exposed as part of the Morpheus system (Jyothi et al. 2016). Morpheus analyzes inter-job data dependencies and ingress/egress operations to automatically derive SLOs. It uses a resource estimator tool, which is also part of Apache Hadoop as of version 2.9, to estimate jobs' resource requirements based on historic runs. Based on the derived SLOs and resource demands, the system generates recurring reservations and submits them for planning. This guarantees that periodic production jobs will have guaranteed access to resources and thus predictable execution.

Further Improvements

In this section, we discuss some additional improvements made to YARN.

Generic resources As more heterogeneous applications with varying resource demands are deployed to YARN clusters, there is an increasing

need for finer control of resource types other than memory and CPU. Examples include disk bandwidth, network I/O, GPUs, and FPGAs.

Adding new resource types in YARN used to be cumbersome, as it required extensive code changes. The upcoming Apache Hadoop 3.1 release (Resource profiles 2017) follows a more flexible resource model, allowing users to add new resources with minimal effort. In fact, users can define their resources in a configuration file, eliminating the need for code changes or recompilation. The Dominant Resource Fairness (Ghodsi et al. 2011) scheduling algorithm at the RM has also been adapted to account for generic resource types, while *resource profiles* can be used for AMs to request containers specifying predefined resource sets. Ongoing work focuses on the isolation of resources such as disk, network, and GPUs.

Node labels Cluster operators can group nodes with similar characteristics, e.g., nodes with public IPs or nodes used for development or testing. Applications can then request containers on nodes with specific labels. This feature is supported by YARN’s Capacity Scheduler from Apache Hadoop 2.6 on (Node labels 2017) and allows at most one label to be specified per node, thus creating nonoverlapping node partitions in the cluster. The cluster administrator can specify the portion of a partition that a queue of the scheduler can access, as well as the portion of a queue’s capacity that is dedicated to a specific node partition. For instance, queue A might be restricted to access no more than 30% of the nodes with public IPs, and 40% of queue A has to be on `dev` machines.

Changing queue configuration Several companies use YARN’s Capacity Scheduler to share clusters across non-coordinating user groups. A hierarchy of queues isolates jobs from each department of the organization. Due to changes in the resource demands of each department, the queue hierarchy or the cluster condition, operators modify the amount of resources assigned to each organization’s queue and to the sub-queues used within that department.

However, queue reconfiguration has two main drawbacks: (1) setting and changing configurations is a tedious process that can only be performed by modifying XML files; (2) queue owners cannot perform any modifications to their sub-queues; the cluster admin must do it on their behalf.

To address these shortcomings, Apache Hadoop 2.9 (OrgQueue 2017) allows configurations to be stored in an in-memory database instead of XML files. It adds a RESTful API to programmatically modify the queues. This has the additional benefit that queues can be dynamically reconfigured by automated services, based on the cluster conditions or on organization-specific criteria. Queue ACLs allow queue owners to perform modifications on their part of the queue structure.

Timeline server Information about current and previous jobs submitted in the cluster is key for debugging, capacity planning, and performance tuning. Most importantly, observing historic data enables us to better understand the cluster and jobs’ behavior in aggregate to holistically improve the system’s operation.

The first incarnation of this effort was the application history server (AHS), which supported only MapReduce jobs. The AHS was superseded by the timeline server (TS), which can deal with generic YARN applications. In its first version, the TS was limited to a single writer and reader that resided at the RM. Its applicability was therefore limited to small clusters.

Apache Hadoop 2.9 includes a major redesign of TS (YARN TS v2 2017), which separates the collection (writes) from the serving (reads) of data, and performs both operations in a distributed manner. This brings several scalability and flexibility improvements.

The new TS collects metrics at various granularities, ranging from *flows* (i.e., sets of YARN applications logically grouped together) to jobs, job attempts, and containers. It also collects cluster-wide data, such as user and queue information, as well as configuration data.

The data collection is performed by collectors that run as services at the RM and at every NM.

The AM of each job publishes data to the collector of the host NM. Similarly, each container pushes data to its local NM collector, while the RM publishes data to its dedicated collector. The readers are separate instances that are dedicated to serving queries via a REST API. By default Apache HBase (Apache HBase 2017) is used as the backing storage, which is known to scale to large amounts of data and read/write operations.

Conclusion

YARN was introduced in Apache Hadoop at the end of 2011 as an effort to break the strong ties between Hadoop and MapReduce and to allow generic applications to be deployed over a common resource management fabric. Since then, YARN has evolved to a fully fledged production-ready resource manager, which has been deployed on shared clusters comprising tens of thousands of machines. It handles applications ranging from batch analytics to streaming and machine learning workloads to low-latency services while achieving high resource utilization and supporting SLOs and predictability for production workloads. YARN enjoys a vital community with hundreds of monthly contributions.

Cross-References

► Hadoop

Acknowledgements The authors would like to thank Subru Krishnan and Carlo Curino for their feedback while preparing this entry. We would also like to thank the diverse community of developers, operators, and users that have contributed to Apache Hadoop YARN since its inception.

References

- Apache Hadoop (2017) Apache Hadoop. <http://hadoop.apache.org>
 Apache HBase (2017) Apache HBase. <http://hbase.apache.org>

- Apache Slider (2017) Apache Slider (incubating). <http://slider.incubator.apache.org>
 Burd R, Sharma H, Sakalanaga S (2017) Lessons learned from scaling YARN to 40 K machines in a multi-tenancy environment. In: DataWorks Summit, San Jose
 Curino C, Difallah DE, Douglas C, Krishnan S, Ramakrishnan R, Rao S (2014) Reservation-based scheduling: if you're late don't blame us! In: ACM symposium on cloud computing (SoCC)
 Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: USENIX symposium on operating systems design and implementation (OSDI)
 Distributed scheduling (2017) Extend YARN to support distributed scheduling. <https://issues.apache.org/jira/browse/YARN-2877>
 Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: fair allocation of multiple resource types. In: USENIX symposium on networked systems design and implementation (NSDI)
 HDFS Federation (2017) Router-based HDFS federation. <https://issues.apache.org/jira/browse/HDFS-10467>
 Jyothi SA, Curino C, Menache I, Narayananurthy SM, Tumanov A, Yaniv J, Mavlyutov R, Goiri I, Krishnan S, Kulkarni J, Rao S (2016) Morpheus: towards automated slos for enterprise clusters. In: USENIX symposium on operating systems design and implementation (OSDI)
 Karanasos K, Rao S, Curino C, Douglas C, Chaliparambil K, Fumarola GM, Heddaya S, Ramakrishnan R, Sakalanaga S (2015) Mercury: hybrid centralized and distributed scheduling in large shared clusters. In: USENIX annual technical conference (USENIX ATC)
 Node Labels (2017) Allow for (admin) labels on nodes and resource-requests. <https://issues.apache.org/jira/browse/YARN-796>
 Opportunistic Scheduling (2017) Scheduling of opportunistic containers through YARN RM. <https://issues.apache.org/jira/browse/YARN-5220>
 OrgQueue (2017) OrgQueue for easy capacityscheduler queue configuration management. <https://issues.apache.org/jira/browse/YARN-5734>
 Placement Constraints (2017) Rich placement constraints in YARN. <https://issues.apache.org/jira/browse/YARN-6592>
 Rasley J, Karanasos K, Kandula S, Fonseca R, Vojnovic M, Rao S (2016) Efficient queue management for cluster scheduling. In: European conference on computer systems (EuroSys)
 Resource Profiles (2017) Extend the YARN resource model for easier resource-type management and profiles. <https://issues.apache.org/jira/browse/YARN-3926>
 Utilization-Based Scheduling (2017) Schedule containers based on utilization of currently allocated containers. <https://issues.apache.org/jira/browse/YARN-1011>
 Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B,

- Baldeschwieler E (2013) Apache Hadoop YARN: yet another resource negotiator. In: ACM symposium on cloud computing (SoCC)
- YARN Federation (2017) Enable YARN RM scale out via federation using multiple RMs. <https://issues.apache.org/jira/browse/YARN-2915>
- YARN JIRA (2017) Apache JIRA issue tracker for YARN. <https://issues.apache.org/jira/browse/YARN>
- YARN TS v2 (2017) YARN timeline service v.2. <https://issues.apache.org/jira/browse/YARN-5355>

ADWIN Algorithm

- ▶ Adaptive Windowing

Alignment Creation

- ▶ Business Process Model Matching

Analytics Benchmarks

Todor Ivanov and Roberto V. Zicari
 Frankfurt Big Data Lab, Goethe University
 Frankfurt, Frankfurt, Germany

Synonyms

[Big data benchmarks](#); [Decision support benchmarks](#); [Machine learning benchmarks](#); [OLAP benchmarks](#); [OLTP benchmarks](#); [Real-time streaming benchmarks](#)

Definitions

The meaning of the word benchmark is (Andersen and Pettersen [1995](#)) *A predefined position, used as a reference point for taking measures against.* There is no clear formal definition of analytics benchmarks.

Jim Gray ([1992](#)) describes the benchmarking as follows: “This quantitative comparison starts

with the definition of a benchmark or workload. The benchmark is run on several different systems, and the performance and price of each system is measured and recorded. Performance is typically a throughput metric (work/second) and price is typically a five-year cost-of-ownership metric. Together, they give a price/performance ratio.” In short, we define that a software benchmark is a program used for comparison of software products/tools executing on a pre-configured hardware environment.

Analytics benchmarks are a type of domain-specific benchmark targeting analytics for databases, transaction processing, and big data systems. Originally, the TPC (Transaction Processing Performance Council) ([TPC 2018](#)) defined online transaction processing (OLTP) (TPC-A and TPC-B) and decision support (DS) benchmarks (TPC-D and TPC-H). The DS systems can be seen as some sort of special online analytical processing (OLAP) system with an example of the TPC-DS benchmark, which is a successor of TPC-H (Nambiar and Poess [2006](#)) and specifies many OLAP and data mining queries, which are the predecessors of the current analytics benchmarks. However, due to the many new emerging data platforms like hybrid transaction/analytical processing (HTAP) (Kemper and Neumann [2011](#); Özcan et al. [2017](#)), distributed parallel processing engines (Sakr et al. [2013](#); Hadoop [2018](#); Spark [2018](#); Flink [2018](#); Carbone et al. [2015](#), etc.), Big data management (AsterixDB [2018](#); Alsubaiee et al. [2014](#)), SQL-on-Hadoop-alike (Abadi et al. [2015](#); Hive [2018](#); Thusoo et al. [2009](#); SparkSQL [2018](#); Armbrust et al. [2015](#); Impala [2018](#); Kornacker et al. [2015](#), etc.), and analytics systems (Hu et al. [2014](#)) integrating machine learning (MLlib [2018](#); Meng et al. [2016](#); MADlib [2018](#); Hellerstein et al. [2012](#)), Deep Learning (Tensorflow [2018](#)) and more, the emerging benchmarks try to follow the trend to stress these new system features. This makes the currently standardized benchmarks (such as TPC-C, TPC-H, etc.) only partially relevant for the emerging big data management systems as they offer new features that require new analytics benchmarks.

Overview

This chapter reviews the evolution of the analytics benchmarks and their current state today (as of 2017). It starts overview of the most relevant benchmarking organizations their benchmark standards and outlines the latest benchmark development and initiatives targeting the emerging Big Data Analytics systems. Last but not least the typical benchmark components are described as well as the different goals that these benchmarks try to achieve.

Historical Background

OLTP and DSS/OLAP

In the end of the 1970s, many businesses started implementing transaction-based systems (Rockart et al. 1982), which later became known as the term online transaction processing (OLTP) systems and represent the instant interaction between the user and the data management system. This type of transaction processing systems became a key part of the companies' operational infrastructure and motivated TPC (TPC 2018) to target these systems in their first formal benchmark specification. At the same time, the decision support systems (DSS) evolved significantly and became a standard tool for the enterprises that assisted in the human decision-making (Shim et al. 2002).

In the early 1990s, a different type of system, called online analytical processing (OLAP) systems by Codd et al. (1993), was used by the enterprises to dynamically manipulate and synthesize historic information. The historic data was aggregated from the OLTP systems, and through the application of dynamic analysis, the users were able to gain important knowledge for the operational activities over longer periods of time.

Over the years, the DS systems were enhanced by the use of the OLAP systems (Shim et al. 2002). They became an essential decision-making tool for the enterprise management and a core element of the company infrastructure. With the wide adaption of multipurpose database

systems to build both an OLTP and DS system, the need for standardized database benchmarks arose. This resulted in an intense competition between database vendors to dominate the market, which leads to the need of domain-specific benchmarks to stress the database software. The use of sample workloads together with a bunch of metrics was not enough to guarantee the product capabilities in a transparent way. Another arising issue was the use of benchmarks for benchmarketing. It happens when a company uses a particular benchmark to highlight the strengths of its product and hide its weaknesses and then promotes the benchmark as a "standard," often without disclosing the details of the benchmark (Gray 1992). All of these opened the gap for standardized benchmarks that are formally specified by recognized expert organizations. Therefore, a growing number of organizations are working on defining and standardizing of benchmarks. They operate as consortia of public and private organizations and define domain-specific benchmarks, price, and performance metrics, measuring and reporting rules as well as formal validation and auditing rules.

TPC

The TPC (Transaction Processing Performance Council) (TPC 2018) is a nonprofit corporation operating as an industry consortium of vendors that define transaction processing, database, and big data system benchmarks. TPC was formed on August 10, 1988, by eight companies convinced by Omri Serlin (TPC 2018). In November 1989 was published the first standard benchmark TPC-A with 42-page specification (Gray 1992). By late 1990, there were 35 member companies. As of 2017, TPC has 21 company members and 3 associate members. There are 6 obsolete benchmarks (TPC-A, TPC-App, TPC-B, TPC-D, TPC-R, and TPC-W), 14 active benchmarks (TPC-C (Raab 1993), TPC-E (Hogan 2009), TPC-H (Pöss and Floyd 2000), TPC-DS (Poess et al. 2017; Pöss et al. 2007; Nambiar and Poess 2006), TPC-DI (Poess et al. 2014), TPC-V (Sethuraman and Taheri 2010), TPCx-HS (Nambiar 2014),

Analytics Benchmarks, Table 1 Active TPC benchmarks (TPC 2018)

Benchmark domain	Specification name
Transaction processing (OLTP)	TPC-C, TPC-E
Decision support (OLAP)	TPC-H, TPC-DS, TPC-DI
Virtualization	TPC-VMS, TPCx-V, TPCx-HCI
Big data	TPCx-HS V1, TPCx-HS V2, TPCx-BB, TPC-DS V2
IoT	TPCx-IoT
Common specifications	TPC-pricing, TPC-energy

Analytics Benchmarks, Table 2 Active SPEC benchmarks (SPEC 2018)

Benchmark domain	Specification name
Cloud	SPEC cloud IaaS 2016
CPU	SPEC CPU2006, SPEC CPU2017
Graphics and workstation performance	SPECapc for solidWorks 2015, SPECapc for siemens NX 9.0 and 10.0, SPECapc for PTC Creo 3.0, SPECapc for 3ds Max 2015, SPECwpc V2.1, SPECviewperf 12.1
High-performance computing, OpenMP, MPI, OpenACC, OpenCL	SPEC OMP2012, SPEC MPI2007, SPEC ACCEL
Java client/server	SPECjvm2008, SPECjms2007, SPECjEnterprise2010, SPECjbb2015
Storage	SPEC SFS2014
Power	SPECpower ssj2008
Virtualization	SPEC VIRT SC 2013

TPCx-BB (Ghazal et al. 2013)), and 2 common specifications (pricing and energy) used across all benchmarks. Table 1 lists the active TPC benchmarks grouped by domain.

SPEC

The SPEC (Standard Performance Evaluation Corporation) (SPEC 2018) is a nonprofit corporation formed to establish, maintain, and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems. It was founded in 1988 by a small number of workstation vendors. The SPEC organization is an umbrella organization that covers four groups (each with their own benchmark suites, rules, and dues structure): the Open Systems Group (OSG), the High-Performance Group (HPG), the Graphics and Workstation Performance Group (GWPG), and the SPEC Research Group (RG). As of 2017, there are around 19 active SPEC benchmarks listed in Table 2.

STAC

The STAC Benchmark Council (STAC 2018) consists of over 300 financial institutions and more than 50 vendor organizations whose purpose is to explore technical challenges and solutions in financial services and to develop technology benchmark standards that are useful to financial organizations. Since 2007, the council is working on benchmarks targeting fast data, big data, and big compute workloads in the finance industry. As of 2017, there are around 11 active benchmarks listed in Table 3.

Other historical benchmark organizations and consortia are The Perfect Club (Gray 1992; Hockney 1996) and the Parkbench Committee (Hockney 1996).

Big Data Technologies

In the recent years, many emerging data technologies have become popular, trying to solve the challenges posed by the new big data and Internet of things application scenarios. In a historical

Analytics Benchmarks, Table 3 Active STAC benchmarks (STAC 2018)

Benchmark domain	Specification name
Feed handlers	STAC-M1
Data distribution	STAC-M2
Tick analytics	STAC-M3
Event processing	STAC-A1
Risk computation	STAC-A2
Backtesting	STAC-A3
Trade execution	STAC-E
Tick-to-trade	STAC-T1
Time sync	STAC-TS
Big data	In-development
Network I/O	STAC-N1, STAC-T0

overview of the trends in data management technologies, Nambiar et al. (2012) highlight the role of big data technologies and how they are currently changing the industry. One such technology is the NoSQL storage engines (Cattell 2011) which relax the ACID (atomicity, consistency, isolation, durability) guarantees but offer faster data access via distributed and fault-tolerant architecture. There are different types of NoSQL engines (key value, column, graph, and documents stores) covering different data representations.

In the meantime, many new big data technologies such as (1) Apache Hadoop (2018) with HDFS and MapReduce; (2) general parallel processing engines like Spark (2018) and Flink (2018); (3) SQL-on-Hadoop systems like Hive (2018) and Spark SQL (2018); (4) real-time stream processing engines like Storm (2018), Spark Streaming (2018) and Flink; and (5) graph engines on top of Hadoop like GraphX (2018) and Flink Gelly (2015) have emerged. All these tools enabled advanced analytical techniques from data science, machine learning, data mining, and deep learning to become common practices in many big data domains. Because of all these analytical techniques, which are currently integrated in many different ways in both traditional database and new big data management systems, it is hard to define the exact features that a successor of the DS/OLAP systems should have.

Big Data Analytics Benchmarks

Following the big data technology trends, many new benchmarks for big data analytics have emerged. Good examples for OLTP benchmarks targeting the NoSQL engines are the Yahoo! Cloud Serving Benchmark (short YCSB), developed by Yahoo, and LinkBench developed by Facebook, described in Table 4. However, most big data benchmarks stress the capabilities of Hadoop as the major big data platform, as listed in Table 5. Others like BigFUN (Pirzadeh et al. 2015) and BigBench (Ghazal et al. 2013) are technology-independent. For example, BigBench (standardized as TPCx-BB) addresses the Big Data 3V's characteristics and relies on workloads which can be implemented by different SQL-on-Hadoop systems and parallel processing engines supporting advanced analytics and machine learning libraries. Since there are no clear boundaries for the analytical capabilities of the new big data systems, it is also hard to formally specify what is an analytics benchmark.

A different type of benchmark, called benchmark suites, has become very popular. Their goal is to package a number of micro-benchmarks or representative domain workloads together and in this way enable the users to easily test the systems for the different functionalities. Some of these suites target one technology like SparkBench (Li et al. 2015; Agrawal et al. 2015), which stresses only Spark, while others like HiBench offer implementations for multiple processing engines. Table 6 lists some popular big data benchmarking suites.

A more detailed overview of the current big data benchmarks is provided in a SPEC Big Data Research Group survey by Ivanov et al. (2015) and a journal publication by Han et al. (2018).

Foundations

In the last 40 years, the OLTP and DS/OLAP systems have been the industry standard systems for data storage and management. Therefore, all popular TPC benchmarks were specified in these areas. The majority TPC benchmark specifications (Poess 2012) have the following main components:

Analytics Benchmarks, Table 4 OLTP benchmarks

Name	Benchmark description
YCSB (Cooper et al. 2010; Patil et al. 2011)	A benchmark designed to compare emerging cloud serving systems like Cassandra, HBase, MongoDB, Riak, and many more, which do not support ACID. It provides a core package of six predefined workloads A–F, which simulate a cloud OLTP application
LinkBench (Armstrong et al. 2013)	A benchmark, developed by Facebook, using synthetic social graph to emulate social graph workload on top of databases such as MySQL and MongoDB

Analytics Benchmarks, Table 5 DS/OLAP/Analytics benchmarks

Name	Benchmark description
MRBench (Kim et al. 2008)	Implementing the TPC-H benchmark queries directly in map and reduce operations
CALDA (Pavlo et al. 2009)	It consists of five tasks defined as SQL queries among which is the original MR Grep task, which is a representative for most real user MapReduce programs
AMP lab big data benchmark (AMPLab 2013)	A benchmark based on CALDA and HiBench, implemented on five SQL-on-Hadoop engines (RedShift, Hive, Stinger/Tez, Shark, and Impala)
BigBench (Ghazal et al. 2013)	An end-to-end big data benchmark that represents a data model simulating the volume, velocity, and variety characteristics of a big data system, together with a synthetic data generator for structured, semi-structured, and unstructured data, consisting of 30 queries
BigFrame (BigFrame 2013)	BigFrame is a benchmark generator offering a benchmarking-as-a-service solution for big data analytics
PRIMEBALL (Ferrarons et al. 2013)	A novel and unified benchmark specification for comparing the parallel processing frameworks in the context of big data applications hosted in the cloud. It is implementation- and technology-agnostic, using a fictional news hub called New York Times, based on a popular real-life news site
BigFUN (Pirzadeh et al. 2015)	It is based on a social network use case with synthetic semi-structured data in JSON format. The benchmark focuses exclusively on micro-operation level and consists of queries with various operations such as simple retrieves, range scans, aggregations, and joins, as well as inserts and updates
BigBench V2 (Ghazal et al. 2017)	BigBench V2 separates from TPC-DS with a simple data model, consisting only of six tables. The new data model still has the variety of structured, semi-structured, and unstructured data as the original BigBench data model. The semi-structured data (weblogs) are generated in JSON logs. New queries replace all the TPC-DS queries and preserve the initial number of 30 queries

- Preamble – Defines the benchmark domain and the high level requirements.
- Database Design – Defines the requirements and restrictions for implementing the database schema.
- Workload – Characterizes the simulated workload.
- ACID – Atomicity, consistency, isolation, and durability requirements.
- Workload scaling – Defines tools and methodology on how to scale the workloads.
- Metric/Execution rules – Defines how to execute the benchmark and how to calculate and derive the metrics.

Analytics Benchmarks, Table 6 Big data benchmark suites

Name	Benchmark description
MRBS (Sangroya et al. 2012)	A comprehensive benchmark suite for evaluating the performance of MapReduce systems in five areas: recommendations, BI (TPC-H), bioinformatics, text processing, and data mining
HiBench (Huang et al. 2010)	A comprehensive benchmark suite consisting of multiple workloads including both synthetic micro-benchmarks and real-world applications. It features several ready-to-use benchmarks from 4 categories: micro benchmarks, Web search, machine learning, and HDFS benchmarks
CloudSuite (Ferdman et al. 2012)	A benchmark suite consisting of both emerging scale-out workloads and traditional benchmarks. The goal of the benchmark suite is to analyze and identify key inefficiencies in the processors core micro-architecture and memory system organization when running todays cloud workloads
CloudRank-D (Luo et al. 2012)	A benchmark suite for evaluating the performance of cloud computing systems running big data applications. The suite consists of 13 representative data analysis tools, which are designed to address a diverse set of workload data and computation characteristics (i.e., data semantics, data models and data sizes, the ratio of the size of data input to that of data output)
BigDataBench (Wang et al. 2014)	An open-source big data benchmark suite consisting of 15 data sets (of different types) and more than 33 workloads. It is a large effort organized in China available with a toolkit that adopts different other benchmarks
SparkBench (Li et al. 2015; Agrawal et al. 2015)	SparkBench, developed by IBM, is a comprehensive Spark-specific benchmark suite that comprises of four main workload categories: machine learning, graph processing, streaming, and SQL queries.

- Benchmark driver – Defines the requirements for implementing the benchmark driver/program.
- Full disclosure report – Defines what needs to be reported and how to organize the disclosure report.
- Audit requirements – Defines the requirements for performing a successful auditing process.

The above structure was typical for the OLTP and DS/OLAP benchmarks defined by TPC, but due to the emerging hybrid OLTP/OLAP systems and big data technologies, these trends have changed (Bog 2013) adapting the new system features. For example, the database schema and ACID properties are not anymore a key requirement in the NoSQL and big data management systems

and are replaced by more general one like system under test (SUT). For example, new categories in TPCx-BB are:

- System under test – Describes the system architecture with its hardware and software components and their configuration requirements.
- Pricing – Defines the pricing of the components in the system under test including the system maintenance.
- Energy – Defines the methodology, rules, and metrics to measure the energy consumption of the system under test in the TPC benchmarks.

The above components are part of the standard TPC benchmark specifications and are not representative for the entire analytics benchmarks

spectrum. Many of the newly defined big data benchmarks are open-source programs. However, the main characteristics of a good domain-specific benchmark are still the same. Jim Gray (1992) defined four important criteria that domain-specific benchmarks must meet:

- Relevant: It must measure the peak performance and price/performance of systems when performing typical operations within that problem domain.
- Portable: It should be easy to implement the benchmark on many different systems and architectures.
- Scalable: The benchmark should apply to small and large computer systems. It should be possible to scale the benchmark up to larger systems and to parallel computer systems as computer performance and architecture evolve.
- The benchmark must be understandable/interpretable; otherwise it will lack credibility.

Similarly, Karl Huppler (2009) outlines five key characteristics that all good benchmarks have:

- Relevant – A reader of the result believes the benchmark reflects something important.
- Repeatable – There is confidence that the benchmark can be run a second time with the same result.
- Fair – All systems and/or software being compared can participate equally.
- Verifiable – There is confidence that the documented result is real.
- Economical – The test sponsors can afford to run the benchmark.

In reality, many of the new benchmarks (in Tables 4, 5 and 6) do not have clear specifications and do not follow the practices defined by Gray (1992) and Huppler (2009) but just provide a workload implementation that can be used in many scenarios. This opens the challenge that the reported benchmark results are not really compa-

rable and strictly depend on the environment in which they were obtained.

In terms of component specification, the situation looks similar. All TPC benchmarks use synthetic data generators, which allow for scalable and deterministic workload generation. However, many new benchmarks use open data sets or real workload traces like BigDataBench (Wang et al. 2014) or a mix between real data and synthetically generated data. This influences also the metrics reported by these benchmarks. They are often not clearly specified or very simplistic (like execution time) and cannot be used for an accurate comparison between different environments.

The ongoing evolution in the big data systems and the data science, machine learning, and deep learning tools and techniques will open many new challenges and questions in the design and specification of standardized analytics benchmarks. There is a growing need for new standardized big data analytics benchmarks and metrics.

Key Applications

The analytics benchmarks can be used for multiple purposes and in different environments. For example, vendors of database-related products can use them to test the features of their data products both in the process of development and after it is released, to position them in the market. The final benchmarking of a data product is usually done by an accredited organization. For example, TPC and SPEC have certified auditors that perform transparent auditing of the complete benchmarking process. The database and big data system administrators can regularly run benchmarks to ensure that the systems are properly configured and perform as expected. Similarly, system architects and application developers use benchmarks to test and compare the performance of different data storage technologies in the process of choosing the best tool for their requirements. Furthermore, benchmarks can be used for different comparisons as in the four categories defined by Jim Gray 1992:

- **To compare different software and hardware systems:** The goal is to use metric reported by the benchmark as a comparable unit for evaluating the performance of different data technologies on different hardware running the same application. This case represents classical competitive situation between hardware vendors.
 - **To compare different software on one machine:** The goal is to use the benchmark to evaluate the performance of two different software products running on the same hardware environment. This case represents classical competitive situation between software vendors.
 - **To compare different machines in a comparable family:** The objective is to compare similar hardware environments by running the same software product and application benchmark on each of them. This case represents a comparison of different generations of vendor hardware or for a case comparing of different hardware vendors.
 - **To compare different releases of a product on one machine:** The objective is to compare different releases of a software product by running benchmark experiments on the same hardware. Ideally the new releases should perform faster (based on the benchmark metric) than its predecessors. This can be also seen as performance regression tests that can assure the new release support all previous system features.
- ▶ [System Under Test](#)
 - ▶ [TPC](#)
 - ▶ [TPC-DS](#)
 - ▶ [TPC-H](#)
 - ▶ [TPCx-HS](#)
 - ▶ [Virtualized Big Data Benchmarks](#)
 - ▶ [YCSB](#)

Cross-References

- ▶ [Auditing](#)
- ▶ [Benchmark Harness](#)
- ▶ [CRUD Benchmarks](#)
- ▶ [Component Benchmark](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [Energy Benchmarking](#)
- ▶ [Graph Benchmarking](#)
- ▶ [Metrics for Big Data Benchmarks](#)
- ▶ [Microbenchmark](#)
- ▶ [SparkBench](#)
- ▶ [Stream Benchmarks](#)

References

- Abadi D, Babu S, Ozcan F, Pandis I (2015) Tutorial: SQL-on-Hadoop systems. *PVLDB* 8(12):2050–2051
- Agrawal D, Butt AR, Doshi K, Larriba-Pey J, Li M, Reiss FR, Raab F, Schiefer B, Suzumura T, Xia Y (2015) SparkBench – a spark performance testing suite. In: TPCTC, pp 26–44
- Alsubaiee S, Altowim Y, Altwaijry H, Behm A, Borkar VR, Bu Y, Carey MJ, Cetindil I, Cheelangi M, Faraaz K, Gabrielova E, Grover R, Heilbron Z, Kim Y, Li C, Li G, Ok JM, Onose N, Pirzadeh P, Tsotras VJ, Vernica R, Wen J, Westmann T (2014) Asterixdb: a scalable, open source BDMS. *PVLDB* 7(14):1905–1916
- AMPLab (2013) <https://amplab.cs.berkeley.edu/benchmark/>
- Andersen B, Pettersen PG (1995) Benchmarking handbook. Chapman & Hall, London
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, Melbourne, 31 May–4 June 2015, pp 1383–1394
- Armstrong TG, Ponnenkanti V, Borthakur D, Callaghan M (2013) Linkbench: a database benchmark based on the Facebook social graph. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2013, New York, 22–27 June 2013, pp 1185–1196
- AsterixDB (2018) <https://asterixdb.apache.org>
- BigFrame (2013) <https://github.com/bigframeteam/BigFrame/wiki>
- Bog A (2013) Benchmarking transaction and analytical processing systems: the creation of a mixed workload benchmark and its application. PhD thesis. <http://d-nb.info/1033231886>
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink™: stream and batch processing in a single engine. *IEEE Data Eng Bull* 38(4):28–38. <http://sites.computer.org/debull/A15dec/p28.pdf>
- Cattell R (2011) Scalable SQL and NoSQL data stores. *SIGMOD Rec* 39(4):12–27
- Codd EF, Codd SB, Salley CT (1993) Providing OLAP (On-line analytical processing) to user-analysis: an IT mandate. White paper

- Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, SoCC 2010, Indianapolis, 10–11 June 2010, pp 143–154
- Ferdman M, Adileh A, Koçberber YO, Volos S, Alisafaei M, Jevdjic D, Kaynak C, Popescu AD, Ailamaki A, Falsafi B (2012) Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: Proceedings of the 17th international conference on architectural support for programming languages and operating systems, ASPLOS, pp 37–48
- Ferrarons J, Adhana M, Colmenares C, Pietrowska S, Bentayeb F, Darmont J (2013) PRIMEBALL: a parallel processing framework benchmark for big data applications in the cloud. In: TPCTC, pp 109–124
- Flink (2018) <https://flink.apache.org/>
- Gelly (2015) <https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html>
- Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen H (2013) Bigbench: towards an industry standard benchmark for big data analytics. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2013, New York, 22–27 June 2013, pp 1197–1208
- Ghazal A, Ivanov T, Kostamaa P, Crolotte A, Voong R, Al-Kateb M, Ghazal W, Zicari RV (2017) Bigbench V2: the new and improved bigbench. In: 33rd IEEE international conference on data engineering, ICDE 2017, San Diego, 19–22 Apr 2017, pp 1225–1236
- GraphX (2018) <https://spark.apache.org/graphx/>
- Gray J (1992) Benchmark handbook: for database and transaction processing systems. Morgan Kaufmann Publishers Inc., San Francisco
- Hadoop (2018) <https://hadoop.apache.org/>
- Han R, John LK, Zhan J (2018) Benchmarking big data systems: a review. *IEEE Trans Serv Comput* 11(3):580–597
- Hellerstein JM, Ré C, Schoppmann F, Wang DZ, Fratkin E, Gorajek A, Ng KS, Welton C, Feng X, Li K, Kumar A (2012) The MADlib analytics library or MAD skills, the SQL. *PVLDB* 5(12):1700–1711
- Hive (2018) <https://hive.apache.org/>
- Hockney RW (1996) The science of computer benchmarking. SIAM, Philadelphia
- Hogan T (2009) Overview of TPC benchmark E: the next generation of OLTP benchmarks. In: Performance evaluation and benchmarking, first TPC technology conference, TPCTC 2009, Lyon, 24–28 Aug 2009, Revised Selected Papers, pp 84–98
- Hu H, Wen Y, Chua T, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. *IEEE Access* 2:652–687
- Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: Workshops proceedings of the 26th IEEE ICDE international conference on data engineering, pp 41–51
- Huppler K (2009) The art of building a good benchmark. In: Nambiar RO, Poess M (eds) Performance evaluation and benchmarking. Springer, Berlin/Heidelberg, pp 18–30
- Impala (2018) <https://impala.apache.org/>
- Ivanov T, Rabl T, Poess M, Queralt A, Poelman J, Poggi N, Buell J (2015) Big data benchmark compendium. In: TPCTC, pp 135–155
- Kemper A, Neumann T (2011) Hyper: a hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: Proceedings of the 27th international conference on data engineering, ICDE 2011, Hannover, 11–16 Apr 2011, pp 195–206
- Kim K, Jeon K, Han H, Kim SG, Jung H, Yeom HY (2008) Mrbench: a benchmark for mapreduce framework. In: 14th international conference on parallel and distributed systems, ICPADS 2008, Melbourne, 8–10 Dec 2008, pp 11–18
- Kornacker M, Behm A, Bittorf V, Bobrovitsky T, Ching C, Choi A, Erickson J, Grund M, Hecht D, Jacobs M, Joshi I, Kuff L, Kumar D, Leblang A, Li N, Pandis I, Robinson H, Rorke D, Rus S, Russell J, Tsirogianis D, Wanderman-Milne S, Yoder M (2015) Impala: a modern, open-source SQL engine for Hadoop. In: CIDR 2015, seventh biennial conference on innovative data systems research, Asilomar, 4–7 Jan 2015, Online proceedings
- Li M, Tan J, Wang Y, Zhang L, Salapura V (2015) SparkBench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM international conference on computing frontiers, pp 53:1–53:8
- Luo C, Zhan J, Jia Z, Wang L, Lu G, Zhang L, Xu C, Sun N (2012) CloudRank-D: benchmarking and ranking cloud computing systems for data processing applications. *Front Comp Sci* 6(4):347–362
- MADlib (2018) <https://madlib.apache.org/>
- Meng X, Bradley JK, Yavuz B, Sparks ER, Venkataraman S, Liu D, Freeman J, Tsai DB, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A (2016) Mllib: machine learning in Apache spark. *J Mach Learn Res* 17:34:1–34:7
- MLLib (2018) <https://spark.apache.org/mllib/>
- Nambiar R (2014) Benchmarking big data systems: introducing TPC express benchmark HS. In: Big data benchmarking – 5th international workshop, WBDB 2014, Potsdam, 5–6 Aug 2014, Revised Selected Papers, pp 24–28
- Nambiar RO, Poess M (2006) The making of TPC-DS. In: Proceedings of the 32nd international conference on very large data bases, Seoul, 12–15 Sept 2006, pp 1049–1058
- Nambiar R, Chitor R, Joshi A (2012) Data management – a look back and a look ahead. In: Specifying big data benchmarks – first workshop, WBDB 2012, San Jose, 8–9 May 2012, and second workshop, WBDB 2012, Pune, 17–18 Dec 2012, Revised Selected Papers, pp 11–19

- Özcan F, Tian Y, Tözün P (2017) Hybrid transactional-/analytical processing: a survey. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD conference 2017, Chicago, 14–19 May 2017, pp 1771–1775
- Patil S, Polte M, Ren K, Tantisiriroj W, Xiao L, López J, Gibson G, Fuchs A, Rinaldi B (2011) YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: ACM symposium on cloud computing in conjunction with SOSP 2011, SOCC’11, Cascais, 26–28 Oct 2011, p 9
- Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M (2009) A comparison of approaches to large-scale data analysis. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2009, Providence, 29 June–2 July 2009, pp 165–178
- Pirzadeh P, Carey MJ, Westmann T (2015) BigFUN: a performance study of big data management system functionality. In: 2015 IEEE international conference on big data, pp 507–514
- Poess M (2012) Tpc’s benchmark development model: making the first industry standard benchmark on big data a success. In: Specifying big data benchmarks – first workshop, WBDB 2012, San Jose, 8–9 May 2012, and second workshop, WBDB 2012, Pune, 17–18 Dec 2012, Revised Selected Papers, pp 1–10
- Poess M, Rabl T, Jacobsen H, Caulfield B (2014) TPC-DI: the first industry benchmark for data integration. *VLDB* 7(13):1367–1378
- Poess M, Rabl T, Jacobsen H (2017) Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. In: Proceedings of the 2017 symposium on cloud computing, SoCC 2017, Santa Clara, 24–27 Sept 2017, pp 573–585
- Pöss M, Floyd C (2000) New TPC benchmarks for decision support and web commerce. *SIGMOD Rec* 29(4):64–71
- Pöss M, Nambiar RO, Walrath D (2007) Why you should run TPC-DS: a workload analysis. In: Proceedings of the 33rd international conference on very large data bases, University of Vienna, 23–27 Sept 2007, pp 1138–1149
- Raab F (1993) TPC-C – the standard benchmark for online transaction processing (OLTP). In: Gray J (ed) The benchmark handbook for database and transaction systems, 2nd edn. Morgan Kaufmann, San Mateo
- Rockart JF, Ball L, Bullen CV (1982) Future role of the information systems executive. *MIS Q* 6(4): 1–14
- Sakr S, Liu A, Fayoumi AG (2013) The family of MapReduce and large-scale data processing systems. *ACM Comput Surv* 46(1):11:1–11:44
- Sangroya A, Serrano D, Bouchenak S (2012) MRBS: towards dependability benchmarking for Hadoop MapReduce. In: Euro-Par: parallel processing workshops, pp 3–12
- Sethuraman P, Taheri HR (2010) TPC-V: a benchmark for evaluating the performance of database applications in virtual environments. In: Performance evaluation, measurement and characterization of complex systems – second TPC technology conference, TPCTC 2010, Singapore, 13–17 Sept 2010. Revised Selected Papers, pp 121–135
- Shim JP, Warkentin M, Courtney JF, Power DJ, Sharda R, Carlsson C (2002) Past, present, and future of decision support technology. *Decis Support Syst* 33(2):111–126
- Spark (2018) <https://spark.apache.org>
- SparkSQL (2018) <https://spark.apache.org/sql/>
- SparkStreaming (2018) <https://spark.apache.org/streaming/>
- SPEC (2018) www.spec.org/
- STAC (2018) www.stacresearch.com/
- Storm (2018) <https://storm.apache.org/>
- Tensorflow (2018) <https://tensorflow.org>
- Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R (2009) Hive – a warehousing solution over a map-reduce framework. *VLDB* 2(2):1626–1629
- TPC (2018) www.tpc.org/
- Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y, Gao W, Jia Z, Shi Y, Zhang S, Zheng C, Lu G, Zhan K, Li X, Qiu B (2014) BigDataBench: a big data benchmark suite from internet services. In: 20th IEEE international symposium on high performance computer architecture, HPCA 2014, pp 488–499

Apache Apex

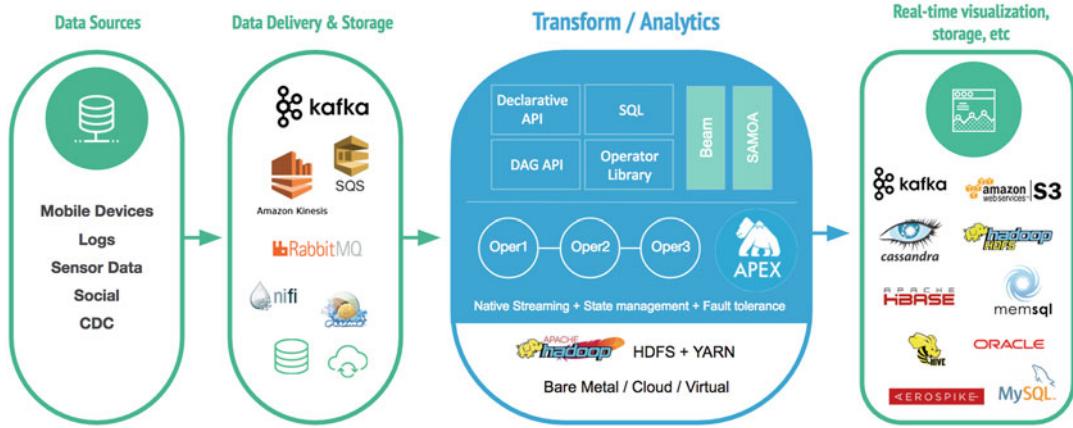
Ananth Gundabattula¹ and Thomas Weise²

¹Commonwealth Bank of Australia, Sydney, NSW, Australia

²Atrato Inc., San Francisco, CA, USA

Introduction

Apache Apex (2018; Weise et al. 2017) is a large-scale stream-first big data processing framework that can be used for low-latency, high-throughput, and fault-tolerant processing of unbounded (or bounded) datasets on clusters. Apex development started in 2012, and it became a project at the Apache Software Foundation in 2015. Apex can be used for real-time and batch processing, based



Apache Apex, Fig. 1 Apex as distributed stream processor

on a unified stateful streaming architecture, with support for event-time windowing and exactly-once processing semantics (Fig. 1).

Application Model and APIs

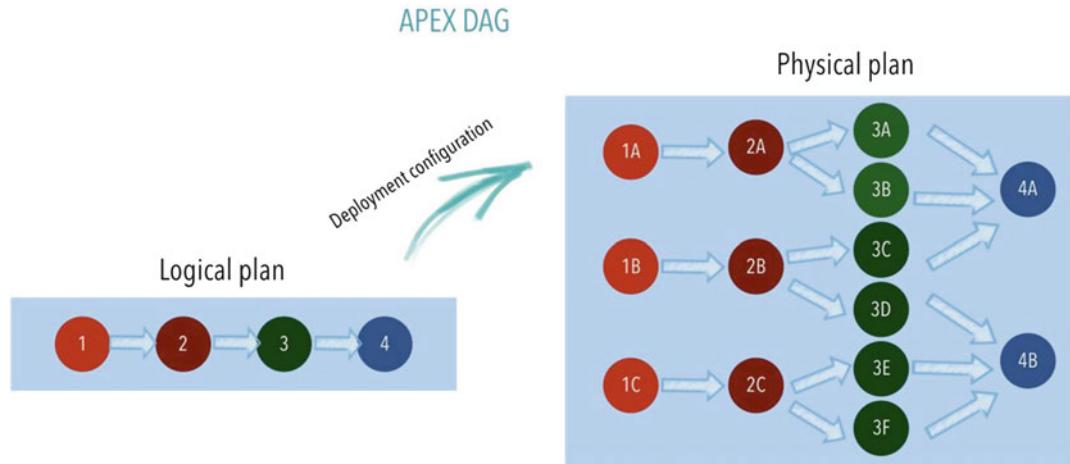
DAG: The processing logic of an Apex application is represented by a directed acyclic graph (DAG) of operators and streams. Streams are unbounded sequences of events (or tuples), and operators are the atomic functional building blocks for sources, sinks, and transformations. With the DAG, arbitrary complex processing logic can be arranged in sequence or in parallel. With an acyclic graph, the output of an operator can only be transmitted to downstream operators. For pipelines that require a loop (or iteration), a special delay operator is supported that allows the output of an operator to be passed back as input to upstream operators (often required in machine learning). The engine also defines a module interface, which can be used to define composite operators that represent a reusable DAG fragment (Fig. 2).

The user-defined DAG is referred to as the logical plan. The Apex engine will expand it into the physical plan, where operators are partitioned (for parallelism) and grouped into containers for deployment. Attributes in the logical plan can

influence these translations, such as the affinity to control which operators should be deployed into the same thread, process, or host.

Low-level, compositional API: The Apex engine defines the low-level API, which can be used to assemble a pipeline by composing operators and streams into a DAG. The API offers a high degree of flexibility and control: Individual operators are directly specified by the developer, and attributes can be used to control details such as resource constraints, affinity, stream encoding, and more. On the other hand, the API tends to be more verbose for those use cases that don't require such flexibility. That's why Apex, as part of the library, offers higher-level constructs, which are based on the DAG API.

High-level, declarative API: The Apex library provides the high-level stream API, which allows the application developer to specify the application through a declarative, fluent style API (similar to API found in Apache Spark and Apache Flink). Instead of identifying individual operators, the developer specifies sources, transformations, and sinks by chaining method calls on the stream interface. The API internally keeps track of operator(s) and streams for eventual expansion into the lower-level DAG. The stream API does not require knowledge of individual operator classes and more concise for use cases that don't require advanced constructs.



Apache Apex, Fig. 2 Apex converts a logical plan into a physical execution model by means of configuration

It is still possible to add customizations as the needs of the application evolve.

SQL: SQL is important for analytics and widely used in the big data ecosystem, with BI tools that can connect to engines such as Apache Hive, Apache Impala, Apache Drill, and others, besides the traditional database systems. SQL isn't limited to querying data; it can also be used to specify transformations. Recent efforts to bring SQL to stream processing like Apache Calcite (2018), Flink (Carbone et al. 2015a), Beam (Akidau et al. 2015), and KSQL (Confluent blog 2018) promise to target a wider audience without lower-level programming expertise, for use cases including ad hoc data exploration, ETL, and more. Apex provides a SQL API that is implemented based on Calcite. It currently covers select, insert, where clause, inner join, and scalar functions. Supported endpoints (read and write) can be file, Kafka, or any other stream defined with the DAG API.

Operators

Operators are Java classes that implement the *Operator* interface and other optional interfaces that are defined in the Apex API and recognized by the engine.

Operators have *ports* (type-safe connection points for the streams) to receive input and emit

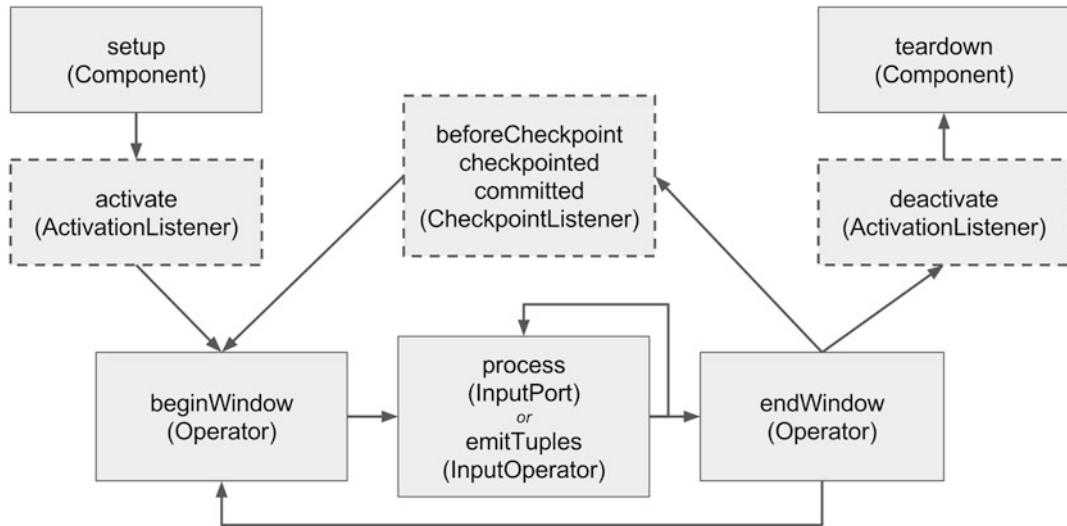
output. Each operator can have multiple ports. Operators that don't have input ports are sources, and operators that don't emit output are sinks. These operators interface with external systems. Operators that have both types of ports typically transform data.

The operator interfaces inform how the engine will interact with the operator at execution time, once it is deployed into a container (Fig. 3).

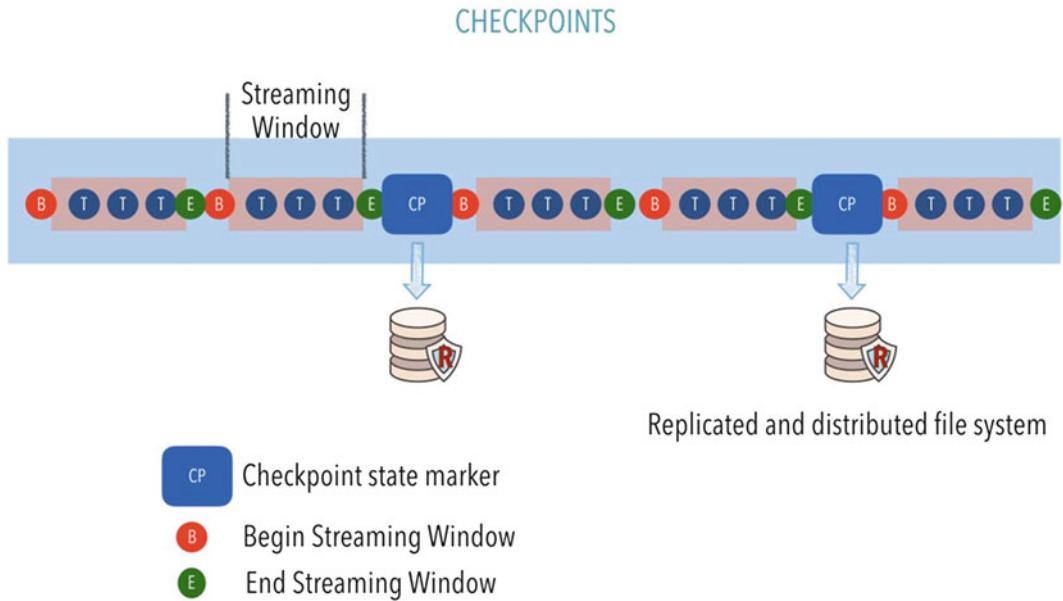
Apex has a continuous operator model; the operator, once deployed, will process data until the pipeline is terminated. The *setup* and *activate* calls can be used for initialization. From then on, data (individual events) will be processed. Periodically the engine will call *begin/endWindow* to demarcate streaming intervals (processing time). This presents an opportunity to perform work that is less suitable for every event. The operator can also implement the optional *CheckpointListener* interface to perform work at checkpoint boundaries.

Checkpointing

Checkpointing in Apex is the foundation for failure handling as well as on-demand parallelism a.k.a. dynamic partitioning. Checkpointing enables application recovery in case of failure by storing the state of the DAG at configurable units



Apache Apex, Fig. 3 Operator lifecycle and interfaces



Apache Apex, Fig. 4 Checkpoint markers injected into the tuple stream at streaming window boundaries

of processing time windows. In case of partitioning (described in the next section), checkpointing allows for resharding the operator states for existing as well as newly spawned operators for elastic scaling (Fig. 4).

Asynchronous and decentralized: Checkpointing is enabled by the framework injecting checkpoint markers into the stream at configured

intervals of ingress/arrival time. An operator instance on the arrival of this marker triggers the checkpointing mechanism. Saving of the state itself is executed asynchronously, without blocking the operator. Completed checkpoints are reported to the application master, which has a view of the entire DAG and can mark a checkpoint as “committed” when it was reported by all operators.

Checkpointing in Apex is thus an asynchronous, lightweight, and decentralized approach.

Idempotency for resumption from checkpoint: Introduction of checkpoint markers into the stream in Apex is a slight variation from Carbone et al. (2015b). The checkpoint markers are introduced at the input operator (source) and traverse the entire DAG. Upon resumption from failure, the stream needs to be replayed in the exact same sequence to enable the entire application as a reproducible state machine. Apex provides the necessary building blocks to achieve idempotency. Individual operators in the DAG ensure idempotency is maintained in their processing logic. For each stream that crosses a container/process boundary, Apex provides a buffer server like (Lin et al. 2016) that can be used by a downstream operator to start consumption from a specific position. This enables fine-grained recovery and dynamic partitioning without a full DAG reset. In case of an input operator, idempotency of replay is either backed by capabilities of the source (in cases like Kafka and files by offset tracking and sequential scan) or the input operator can record the source data for replay by using a write-ahead log (WAL) implementation that is provided by the Apex library.

Opt-in independence: Operators in Apex can choose to opt out of checkpointing cycles to avoid serialization overhead. As an example, an operator that is representing a matrix transpose operation can choose to not be part of the checkpointing process as it does not need a state to be accumulated across tuples or across streaming windows. On the other hand, an operator which is computing a cumulative sum or part of sort, join logic in SQL DAG needs the state to be handed from one streaming window to another and hence needs to be stateful. Stateless operators can be enabled either by an annotation marker on the operator implementation or declaring it in the configuration provided as input to the application launch. For stateful operators, Apex ensures that the state is serialized at checkpoint intervals. The default serialization mechanism allows the operator implementation to decide (and optimize) which fields are serialized (or skipped by marking

them transient) and also how fields are serialized through optional annotations.

Exactly-once state and exactly-once processing: Exactly-once processing is the holy grail of distributed stream processing engines. In reality though, reprocessing cannot be avoided in a distributed system, and therefore exactly-once refers to the effect on the state, which is also why it is alternatively referred to as “effectively once.” While some engines define exactly-once semantics purely from internal state persistence point of view, others aim to provide constructs for exactly-once processing in addition to state with varying degrees of implementation complexity as given in Kulkarni et al. (2015), Noghabi et al. (2017), and Jacques-Silva et al. (2016). Exactly-once state management in Apex is a distributed process by the virtue of each operator triggering its own state passivation when the checkpoint tuple is processed. While some streaming systems only focus on exactly-once semantics for internal state, the Apex library provides support for end-to-end exactly-once for many of its connectors, thereby also covering the effect of processing on the state in respective external systems. This is possible through interface contracts in the low-level API and idempotency and by utilizing capabilities that are specific to the integrated system, such as transactions, atomic renames, etc.

At-most-once and at-least-once processing semantics: It is not atypical for at-least-once processing semantics for use cases like finding a max value in a given stream, while at-most-once processing is required for use cases where recency of data processing state matters the most. At-least-once semantics is achieved by the upstream operator replaying the tuples from the checkpoint and the downstream operator not implementing any lightweight checkpoint state-based checks. For at-most-once semantics, the upstream operator needs to just stream tuples to downstream without starting from a checkpoint.

Low-level API for processing semantics: As described in the Operator section, the operator is given a chance to perform business logic-specific process at streaming window boundaries as well as checkpoint boundaries. Several operators in Apex implement incremental lightweight state

saving at window boundaries and a complete operator state at checkpointing boundaries. Utilities exist in the framework to get/set data structures that represent state using a window ID marker as input thus allowing for a lightweight state passivation. Since a checkpoint resumption can result in reprocessing multiple windows, this incremental state can come in handy to completely skip windows that are processed from a certain checkpoint. Let us consider how this approach provides for exactly-once processing semantics in systems that do not provide two-phase commit transaction support. Until the last streaming window before a crash (referred to as orphaned window hereafter), state can be passivated incrementally at streaming window boundaries. For the orphaned window, for which neither a lightweight state has been persisted nor a full checkpoint cycle is complete, custom business logic can be invoked using a “check/read and then process” pattern. This “check and process along with a business helper function” is only executed for the orphaned window reprocessing. Processing resumes a normal pattern beyond this orphaned window processing. Thus a combination of lightweight checkpointing at window boundaries, full checkpointing at checkpoint boundaries, and the check and process pattern can achieve exactly-once processing semantics in Apex for this class of systems.

Managed state and spillable data structures: State saving at checkpointing boundary may be sub-optimal when the state is really large (Del Monte 2017). Incremental state persistence at smaller intervals and/or spillable state can be effective patterns to mitigate this (To et al. 2017; Fernandez et al. 2013; Akidau et al. 2013; Sebepou and Magoutis 2011). Apex supports both. To handle very large state, the Apex library provides for spillable data structures similar to Carbone et al. (2017). Instead of a full copy local store like RocksDB, spillable data structures in Apex are block structured and by default backed by the distributed file system (DFS), allowing for delta writes and on-demand load (from DFS) when operators are restored.

Flexibility, ease of use, and efficiency have been core design principles for Apex, and checkpointing is no exception. When desired, the user

has complete control to define the data structures that represent the operator state. For recovery, Apex will only reset to checkpoint the part of the DAG that is actually affected by a failure. These are examples of features that set Apex apart from other streaming frameworks like Zaharia et al. (2012) and Carbone et al. (2015b). With flexibility at the lower level, higher-level abstractions are provided as part of the library that optimize for specific use cases.

High Availability

High availability in Apex is achieved by extending two primary constructs. Leverage YARN as cluster manager to handle process and node failure scenarios and use the distributed file system to recover the state. An Apex application is associated with an application ID and is either given a new ID or resume from a previous application identity as identified by an ID given as a startup parameter. Apex being a YARN native application utilizes the YARN Resource manager to allocate the application master (AM) at the time of launch. AM either instantiates a new physical deployment plan using the logical plan or reuses an existing physical plan if an existing application ID is passed at startup. As the DAG is physicalized and starts executing, each of the JVM operators sends heartbeats to the AM using a separate thread. Subsequent to this, there are many scenarios that can result in a fault scenario and recovery options possible as given in Nasir (2016).

Worker container failure: In case of a container JVM crashing or stalling for a long time due to a GC pause, AM detects the absence of the heartbeat and initiates a kill (if applicable) and redeploy sequence. AM would negotiate with YARN for the replacement container and request to kill the old container (if it was stalled). The new instance would then resume from a checkpointed state. It may be noted that the entire sub-DAG downstream to the failed operator will be redeployed as well to ensure the semantics of processing are upheld especially for cases when the

downstream operators are implementing exactly-once semantics.

AM failure: During an AM crash, the individual worker containers continue processing the tuples albeit without any process taking care of the recovery and monitoring. In the meantime, YARN would detect that AM container has gone down and redeploy a new instance of it and pass the previous application ID as part of the restart process. This new instance would then resume from its checkpointed state. The checkpointed state of AM is more related to the execution state of the DAG. This new AM container would update its metadata in the distributed file system representing the application ID which in turn would be picked up by the worker containers to reestablish their heartbeat cycles.

Machine failures: In the event of a physical node failure, YARN resource manager (RM) would be notified of the node manager (NM) death. This would result in RM deploying all containers currently running on that NM. It is possible that there are multiple containers of the application on the failed host, and all of these would be migrated to new host(s) using the process described before.

Resource Management

Apex is a YARN native application that follows the YARN principles of resource request and grant model. The application master upon launch generates a physical execution plan from the DAG logical representation. The application master itself is a YARN container. The application master then spawns the operator instances by negotiating operator containers from the YARN resource manager, taking into account

the memory and compute resource settings in the DAG (Fig. 5).

Deployment models: The Apex operator deployment model allows for four different patterns:

1. Thread – Operator logic is invoked by a single thread in the same JVM.
2. Container – Operators coexist in the same JVM.
3. Host – Operators are available on the same node.
4. Default – Highest cost in terms of serialization and IPC.

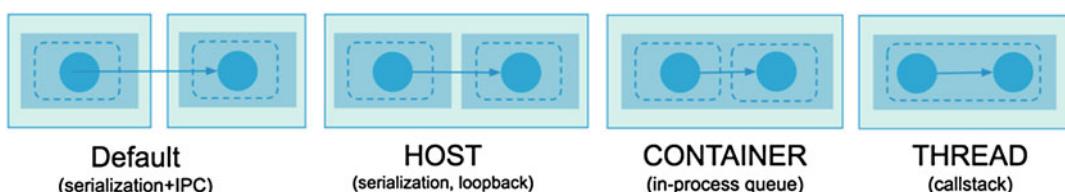
Affinity or anti-affinity patterns further allow for customized location preferences while deploying the operators by the application master. Affinity allows for multiple operators to be located on the same host or share the same container or thread.

Partitioning/Scaling

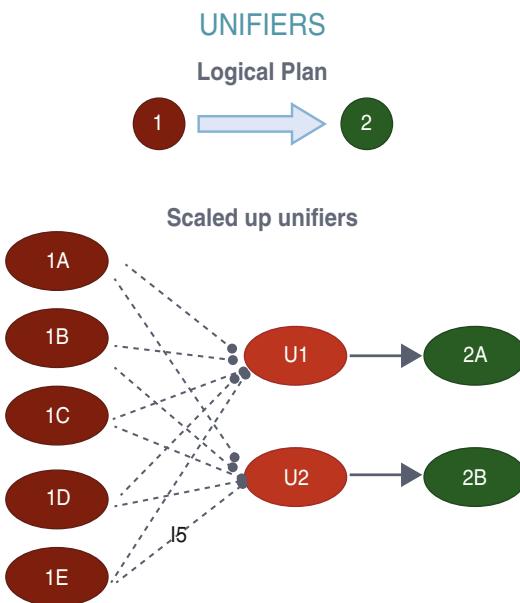
Operator parallelism a.k.a partitioning helps in dealing with latency and throughput trade-off aspects of a streaming application design. Apex enables developers to concentrate on the business logic and enable a configuration-based approach to scale a logical implementation to a scalable physical deployment model. This approach is referred to as partitioning and is enabled by configuring a partitioner for an operator.

Partitioners

A partitioner implementation can be specified how to partition the operator instances. While use case like specifying a fixed number of partitions



Apache Apex, Fig. 5 Operator deployment models



Apache Apex, Fig. 6 Unifiers allow for varying parallelism between operators. U1 and U2 unifiers are automatically injected into the DAG at deployment time

at startup can be met by using one of the partitioners available as part of the framework, Apex allows for custom implementation as well.

Such custom implementations can cater to many use cases like deciding partitions on the input source system characteristics. As an example, the Kafka partitioners as part of the Apex library have the capability to scale to as many Kafka partitions that each of the configured Kafka topics have or alternatively support mapping multiple Kafka partitions to one Apex operator partition. A partitioner can be stateful; it can re-shard the state of the old partitions (Fig. 6).

Unifiers

While partitioning allows for scalability of the operator business logic, it invariably results in an impedance mismatch if the downstream operator is not at the right scaling factor. Apex allows parallelism to vary between operators. In such cases, unifiers are automatically injected into the physical plan and define how the upstream results are shuffled to the new parallelism level. The user can customize the unifier implementation, which

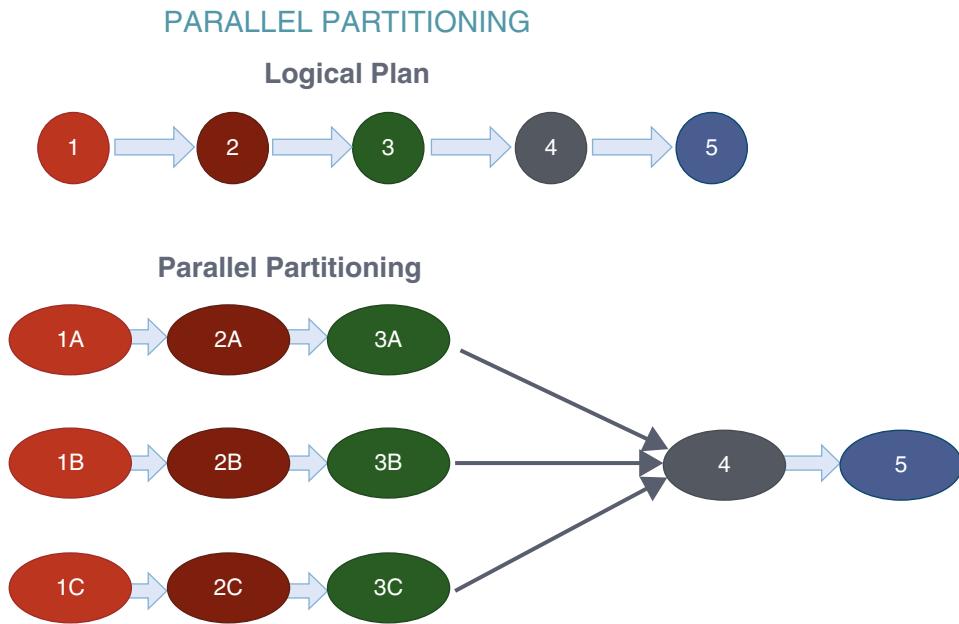
typically depends on the operator logic. A unifier is an operator without input port that is set on the output port of the partitioned operator. Cascading unifier patterns can also be implemented wherein the partitioned operator output can itself be shuffled in stages before streaming the merged result to the downstream operator. For a reduce operation, this can be used to overcome resource limits (network, CPU, etc.) for a latency trade-off (Fig. 7).

Parallel Partitioning

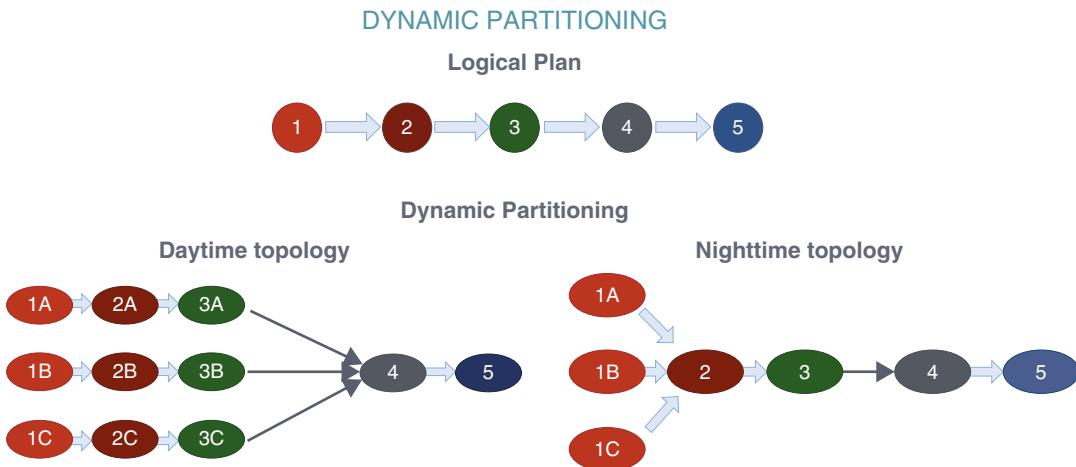
While partitioners and unifiers allow for independent scaling of each operator of an application, there will be use cases where downstream operators can align with the upstream operators' level of parallelism and avoid a unifier, thus further decreasing the overall latencies due to decreased network hops (in some systems, this also referred to as chaining). This is enabled via a configuration parameter set on the downstream operator, and this parallel partitioning can be configured for as many downstream operators as the application design mandates it to be (Fig. 8).

Dynamic Partitioning

Load variation is a common pattern across many streaming systems, and some of these engines provide for dynamic scaling as given in Floratou et al. (2017) and Bertolucci et al. (2015). Also the advent of cloud computing where cost is based on resource consumption models as given in Hummer et al. (2013) and Sattler and Beier (2013) makes compelling case for the need to dynamically adjust resources based on context. Static partitioning may not be sufficient to achieve latency SLAs or optimize resource consumption. Apex allows partitions to be scaled or contracted dynamically at checkpointing boundaries. A partitioner implementation can use the operational metrics of the physical operator instances to decide on the optimal scaling configuration. Unlike other streaming engines like Spark which need stand-alone shuffle service, Apex dynamic partitioning can be aligned with the application patterns.



Apache Apex, Fig. 7 Parallel partitioning can avoid shuffling overheads when it is not necessary



Apache Apex, Fig. 8 Dynamic partitioning allows for scale-up and scale-down strategies at runtime. Example above has different topologies at daytime and nighttime allowing for efficient use of hardware

Integration Using Apex Library

Apex enables a faster time to market model by shipping many connectors to external systems that can act as a source or a sink or both. The Apex library is also referred to as Malhar. Some of the common integrations include JDBC driver-enabled databases like Oracle, MySQL, and Postgres; replayable sources like Kafka and files;

NOSQL databases like Cassandra and HBase; and JMS-enabled systems like MQ besides many others. The library implementations provide value add by not only implementing read/write patterns to these systems but also aiming to provide exactly-once processing.

File systems: Apex operators bring in lot more to the maturity of the implementation by enabling enterprise patterns like polling directories for new

file arrivals, offset tracking backed sequential scanning approach, resumption from checkpoints in case of failures, handling of varied formats of the file contents, and compression formats while writing contents and even splitting very large files into blocks while reading to enable efficient partitioning strategies.

Kafka: Kafka integration is one of the battle-tested integrations of Apex. Kafka operators allow for flexible mapping of operators wherein the mapping configuration can map “m” Kafka partitions across “n” Apex operators. It may be noted that m can represent partitions that may span across multiple Kafka clusters and multiple Kafka topics spread across these Kafka topics. Apex Kafka operators also support exactly-once semantics for Kafka versions prior to 0.11 version thus taking the burden of a transaction to Apex processing layer as opposed to relying on Kafka. The integration in Apex thus can be considered far richer as compared to the other systems.

Non-replayable sources like JMS: Exactly-once semantics at input sources is a bit more involved in case of systems like JMS wherein the source does not have a contract to replay a data point once a handover handshake is complete. Apex provides for a WAL implementation which is used by the JMS operator to replay a series of previously acknowledged messages when restarting from a checkpoint.

JDBC-enabled databases: JDBC-enabled databases like Oracle, Postgres, and MySQL allow for an exactly-once write pattern through transactions. The Apex connector can use this to commit transactions at streaming windows or checkpoint boundaries. The JDBC source supports idempotent read by tracking the offset (for a query with order by clause).

Conclusion

Apex is an enterprise-grade distributed streaming engine that comes with many foundational constructs as well as connectors that help in faster time to market as compared to some similar systems. Dockerized containers and support for next-generation container orchestration are some

of the features that would benefit Apex in the near future.

References

- Akida T et al (2013) MillWheel: fault-tolerant stream processing at internet scale. *PVLDB* 6:1033–1044
- Akida T et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *PVLDB* 8:1792–1803
- Apache Apex (2018) <https://apex.apache.org/>
- Apache Calcite (2018) <https://calcite.apache.org/>
- Bertolucci M et al (2015) Static and dynamic big data partitioning on Apache Spark. *PARCO*
- Carbone P et al (2015a) Apache Flink™: stream and batch processing in a single engine. *IEEE Data Eng Bull* 38:28–38
- Carbone P et al (2015b) Lightweight asynchronous snapshots for distributed dataflows. *CoRR* abs/1506.08603: n. pag
- Carbone P et al (2017) State management in Apache Flink®: consistent stateful distributed stream processing. *PVLDB* 10:1718–1729
- Confluent blog (2018) <https://www.confluent.io/blog/ksql-open-source-streaming-sql-for-apache-kafka/>
- Del Monte B (2017) Efficient migration of very large distributed state for scalable stream processing. *PhD@VLDB*
- Fernandez RC et al (2013) Integrating scale out and fault tolerance in stream processing using operator state management. *SIGMOD conference*
- Floratou A et al (2017) Dhalion: self-regulating stream processing in Heron. *PVLDB* 10:1825–1836
- Hummer W et al (2013) Elastic stream processing in the cloud. *Wiley Interdisc Rev: Data Min Knowl Discov* 3:333–345
- Jacques-Silva G et al (2016) Consistent regions: guaranteed tuple processing in IBM streams. *PVLDB* 9: 1341–1352
- Kulkarni S et al (2015) Twitter Heron: stream processing at scale. *SIGMOD conference*
- Lin W et al (2016) StreamScope: continuous reliable distributed processing of big data streams. *NSDI*
- Nasir MAU (2016) Fault tolerance for stream processing engines. *CoRR* abs/1605.00928: n. pag
- Noghabi SA et al (2017) Stateful scalable stream processing at LinkedIn. *PVLDB* 10:1634–1645
- Sattler K-U, Beier F (2013) Towards elastic stream processing: patterns and infrastructure. *BD3@VLDB*
- Sebepou Z, Magoutis K (2011) CEC: continuous eventual checkpointing for data stream processing operators. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). pp 145–156
- To Q-C et al (2017) A survey of state management in big data processing systems. *CoRR* abs/1702.01596: n. pag
- Weise T et al (2017) Learning Apache Apex. *Packt Publishing*

- Zaharia M et al (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI
- Zaharia M et al (2013) Discretized streams: fault-tolerant streaming computation at scale. SOSP

Apache Flink

Fabian Hueske and Timo Walther
Data Artisans GmbH, Berlin, Germany

Synonyms

[Stratosphere platform](#)

Definitions

Apache Flink is a system for distributed batch and stream processing. It addresses many challenges related to the processing of bounded and unbounded data. Among other things, Flink provides flexible windowing support, exactly-once state consistency, event-time semantics, and stateful stream processing. It offers abstractions for complex event processing and continuous queries.

Overview

Today, virtually all data is continuously generated as streams of events. This includes business transactions, interactions with web or mobile application, sensor or device logs, and database modifications. There are two ways to process continuously produced data, namely batch and stream processing. For stream processing, the data is immediately ingested and processed by a continuously running application as it arrives. For batch processing, the data is first recorded and persisted in a storage system, such as a file system or database system, before it is (periodically) processed by an application that processes a bounded data set. While stream processing

typically achieves lower latencies to produce results, it induces operational challenges because streaming applications which run 24×7 make high demands on failure recovery and consistency guarantees.

The most fundamental difference between batch and stream processing applications is that stream processing applications process continuously arriving data. The two core building blocks when processing streaming data are state and time. Applications require state for every non-trivial computation that involves more than a single event. For example, state is required to collect multiple events before performing a computation or to hold the result of partial computations. Time on the other hand is important to determine when all relevant data was received and a computation can be performed. Having good control over time enables applications to treat result completeness for latency. Time is not relevant in batch processing because all input data is known and present when the processing starts.

Stateful stream processing is a very versatile architectural pattern that can be applied to a wide spectrum of data-related use cases. Besides providing better latency than batch processing solutions, stateful stream processing applications can also address use cases that are not suitable for batch processing approaches at all. Event-driven applications are stateful stream processing applications that ingest continuous streams of events, apply business logic to them, and emit new events or trigger external actions. These applications share more characteristics with transactional workloads than analytical applications. Another common use case for stateful stream processing is complex event processing (CEP), which is applied to evaluate patterns over event streams.

Historical Background

Apache Flink originates from an academic research project. In 2010, the Stratosphere project was started by five research groups from Technische Universität Berlin, Humboldt

Universität zu Berlin, and Hasso Plattner Institute Potsdam (<http://gepris.dfg.de/gepris/projekt/132320961?language=en>. Visited on 22 Dec 2017). The goal of the project was to develop novel approaches for large-scale distributed data processing. In the course of the project, the researchers developed the prototype of a data processing system to evaluate the new approaches and released the software as open source under the Apache software license (Alexandrov et al. 2014).

When the project started in 2010, the Apache Hadoop project (<https://hadoop.apache.org>. Visited on 22 Dec 2017), an open-source implementation of Google’s MapReduce (Dean and Ghemawat 2008) and GFS (Ghemawat et al. 2003) publications, had gained a lot of interest in research and industry. MapReduce’s strong points were its ability to scale data processing tasks to a large number of commodity machines and its excellent tolerance for hardware and software failures. However, the database research community had also realized that MapReduce was neither the most user-friendly nor most efficient approach to define complex data analysis applications. Therefore, the Stratosphere project aimed to build a system that combined the advantages of MapReduce and relational database systems. The first result was a system consisting of the PACT programming model, the distributed dataflow processing engine Nephele, and an optimizer that translated PACT programs into Nephele dataflows (Battre et al. 2010). The PACT programming model generalized the MapReduce programming model by providing more parallelizable operator primitives and defining programs as directed acyclic dataflows. Hence, specifying complex analytical applications became much easier (Alexandrov et al. 2011). The runtime operators to execute PACT programs were implemented based on well-known algorithms from database system literature, such as external merge-sort, block-nested loop join, hybrid-hash join, and sort-merge join. Having a choice in runtime operators and data distribution strategies due to the flexibility of the distributed dataflow execution engine Nephele resulted in different alternatives of how a PACT program could be executed. Similar to physical optimization in re-

lational database systems, a cost-based optimizer enumerated execution plans under consideration of interesting and existing physical properties (such as partitioning, sorting, and grouping) and chose the least expensive plan for execution. Compared to the MapReduce programming and execution model, the Stratosphere research prototype provided a programming API that was as versatile as MapReduce but eased the definition of advanced data analysis applications, an efficient runtime based on concepts and algorithms of relational database systems, and a database-style optimizer to automatically choose efficient execution plans. At the same time, the prototype had similar hardware requirements and offered similar scalability as MapReduce. Based on the initial Stratosphere prototype, further research was conducted on leveraging static code analysis of user-defined function for logical program optimization (Hueske et al. 2012) and the definition and execution of iterative programs to support machine learning and graph analysis applications (Ewen et al. 2012).

In May 2014, the developers of the Stratosphere prototype decided to donate the source code of the prototype to the Apache Software Foundation (ASF) (<https://wiki.apache.org/incubator/StratosphereProposal>. Visited on 22 Dec 2017). Due to trademark issues, the project was renamed to Apache Flink. Flink is the German word for “nimble” or “swift.” The initial group of committers consisted of the eight Stratosphere contributors and six members of the Apache Incubator to teach the Flink community the Apache Way. In August 2014, version 0.6 of Apache Flink was released as the first release under the new name Apache Flink (<http://flink.apache.org/news/2014/08/26/release-0.6.html>. Visited on 22 Dec 2017). Three months later in November 2014, version 0.7 was released. This release included a first version of Flink’s DataStream API. With this addition, Flink was able to address stream as well as batch processing use cases with a single processing engine (Carbone et al. 2015a). Because the distributed dataflow processor (formerly known as Nephele) had always supported pipelined data transfers, the new stream processing capabilities did not require major changes

to the engine. In January 2015, 9 months after the start of the incubation, Flink left the incubator and became a top-level project of the Apache Software Foundation (https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces69). Visited 22 Dec 2017).

While until Flink’s graduation the community was mostly working on batch processing features, stream processing slowly became the new focus of the community. Over the next couple of releases, features such as windowing support, exactly-once state consistency, event-time semantics, stateful stream processing, and high availability for worker and master processes were added. Moreover, the DataStream API was declared stable and support for persisting application state in savepoints, and restarting applications from savepoints as well as maintaining very large operator state in RocksDB were implemented. When version 1.0.0 was released in March 2016 (<https://flink.apache.org/news/2016/03/08/release-1.0.0.html>. Visited on 22 Dec 2017), Flink had become a fully fleshed stream processor with a feature set that was unique among other open-source stream processors.

Since Flink’s 1.0.0 release until today (December 2017, version 1.4.0), many more new significant features were added, such as SQL support for unified batch and streaming queries (<https://flink.apache.org/news/2017/04/04/dynamic-tables.html>. Visited on 22 Dec 2017), a complex event processing (CEP) library to identify and react on patterns in event streams (<https://data-artisans.com/blog/complex-event-processing-flink-cep-update>. Visited 22 Dec 2017), support for scaling applications in and out (Carbone et al. 2017), and support for querying operator state from external applications. By now, the Flink community has grown to 34 committers, and more than 350 individuals have contributed to Flink. Apache Flink is used in production at very large scale by enterprises around the world and across various industries, such as Alibaba, DellEMC, ING, King, Netflix, and Uber. Some of these users have built services for internal users or even expose these services to paying customers.

Foundations

Apache Flink is a system for batch and stream processing use cases (Carbone et al. 2015b). The main APIs, namely, the *DataSet API* for batch and *DataStream API* for streaming programs, allow to fluently specify a data processing plan by using first-order and second-order functions known from functional programming. *Second-order functions* give certain guarantees about the distributed execution. *First-order functions* implement custom business logic within the provided guarantees. For example, the map operator guarantees to apply a first-order function to every record, and keyBy partitions and distributes the stream by using a key specified in the first-order function. In the following, the discussion focuses on the DataStream API and Flink’s stream processing capabilities since this reflects the current evolution of the system and the majority of its production use cases.

Figure 1 shows an example of a Flink DataStream API program that reads from a publish-subscribe messaging system. The program transforms the input by applying a user-defined map function to every record. Afterward, it partitions the records by a key and aggregates multiple records over a window of 5 seconds that discretizes the stream. The result is written into a continuous single log file. Flink automatically gathers information about the return type of each operator and generates appropriate record serializers for the network shipping. The API calls only construct a logical representation and are translated into a directed acyclic job graph that is submitted to the cluster for execution.

Figure 2 illustrates the distributed execution of the example program. A Flink setup consists of two types of processes: the *JobManager* and the *TaskManagers*. For high availability, there might be multiple standby JobManagers. The JobManager is the master that is responsible for cluster coordination, metrics collection, and monitoring. It receives a job graph and constructs an execution graph with knowledge of the available TaskManagers and their allocation. The execution graph is deployed on the TaskManagers. The TaskManagers are the workers that are

```

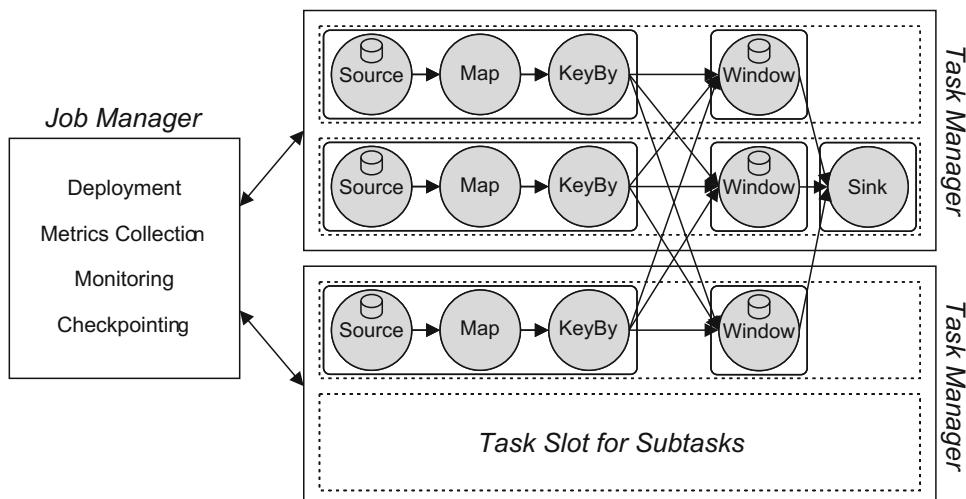
// setup the environment
StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();

// define a streaming pipeline
env.addSource(new FlinkKafkaConsumer011<>(...))
    .map(new UserDefinedMapFunction())
    .keyBy("userId")
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
    .apply(new UserDefinedWindowFunction())
    .writeAsText("/file.log").setParallelism(1);

// execute the constructed plan
env.execute();

```

Apache Flink, Fig. 1 Apache Flink DataStream API program



Apache Flink, Fig. 2 Flink distributed execution model

responsible for executing operator pipelines and exchanging data streams.

The example from Fig. 2 is executed with a degree of parallelism equal to 3 and thus occupies three task slots. *Task slots* split the resources of a TaskManager into a finer granularity and define how many slices of a program can be executed concurrently by a TaskManager; the dotted lines

in Fig. 2 highlight the task slots. Each pipeline is independently evaluated. Once an operator's result is ready, it will be forwarded immediately to the next operator without additional synchronization steps. A pipeline consists of one or more subtasks. Depending on the operations, a *subtask* executes only one operator, such as the window operator, or multiple operators that are

concatenated by so-called operator chaining. In the example, the reading of a record from the source, its transformation, and the extraction of a key are performed in a chain without having to serialize intermediate data. Both the `keyBy` and the sink operation break the chain, because `keyBy` requires a shuffling step and the sink has a lower parallelism than its predecessor.

Operators in Flink DataStream programs can be stateful. In our example, the sources need to store the current read offset in the log of the publish-subscribe system to remember which records have been consumed so far. Window operators need to buffer records until the final aggregation at the end of a window can be triggered. The JobManager is responsible for coordinating the creation of consistent checkpoints of a program's state, i.e., the state of all operators, and restarting the program in case of failures by loading the latest complete checkpoint and replaying parts of a stream. Flink's checkpoint and recovery mechanism will be covered later in more detail.

Time Handling

The handling of *time* receives special attention in Flink's DataStream API. Most streaming applications need a notion of time to define operations on conceptually never-ending inputs. The previous example included a *tumbling time window* to discretize the stream into segments containing the records received within an interval of 5 seconds. Similarly, *sliding windows* allow for similar fixed-length windows but with a possible overlap, e.g., a window of 1 minute length that is evaluated every 10 seconds. *Session windows* have a variable length and are evaluated depending on a gap of inactivity. These window types as well as several other operations on data streams depend on a notion of time to specify an interval of the stream that they interact with. Time-based operators need to be able to look up the “current time” while processing a record of a stream. Flink supports two time modes.

Processing time is defined by the local clock of the machine that processes an operator. Perform-

ing operations based on the wall-clock time is the easiest notion of time with little overhead. However, it assumes a perfect environment where all records arrive in a strict order, can be processed on time without any side effects, and do not need to be reprocessed. In the real world, events might arrive in the wrong order, or large amounts of events might arrive at the same time and, thus, logically belong to the same window but do not reach the window operator punctually. Moreover, the amount of hardware resource available for processing can affect which records are grouped together. Hence, processing time is not applicable if quality-of-service specifications require that results, which are computed from a log of events, and must be reproducible.

Event time addresses the issues above by relying on a timestamp that is associated with every record and by defining a strategy to make progress that guarantees consistent results. Flink adopts the *dataflow model* (Akidau et al. 2015). In addition to timestamped records, data streams need to be enriched with *watermarks* in order to leverage event time. Watermarks are metadata records with a timestamp that indicate that no record with a timestamp lower than the watermark's timestamp will be received from a connection in the future. Every operator in Flink contains logic to compute a new watermark from the watermarks it receives via its incoming connections. The watermark of an operator acts as its internal clock and triggers computations, such as the computation of a window when the watermark passes its end time. An operator tracks for each incoming connection the highest observed watermark and computes its own watermark as the smallest watermark across the current watermarks of its incoming connections. Whenever an operator advances its own watermark, it performs all computations that are triggered by the new watermark, emits the resulting records, and subsequently broadcasts its new watermark across all its outgoing connections. This mechanism ensures that watermarks are strictly increasing and that emitted records remain aligned with the emitted watermarks.

State and Fault Tolerance

The ability to memorize information that was received or computed for future computations is another crucial feature in streaming applications. *State* can be used to buffer records until a certain event occurs, to maintain (partial) aggregates based on incoming events, or to train and store machine learning models. From a user’s point of view, state can be considered as a set of member variables in each operator. A member variable is empty at the beginning and can be accessed and modified depending on its data type. Since the values in state variables affect the computed result, state must be resilient to failures, recoverable, and flexible enough for rescaling.

Flink takes care of snapshotting the content of state variables by performing so-called Checkpoints in regular intervals (e.g., every minute). Flink employs *lightweight asynchronous snapshots* (Carbone et al. 2015c) based on the *Chandy-Lamport algorithm* (Mani Chandy and Lamport 1985) for distributed snapshots. When a snapshot is triggered, *checkpoint barriers* with a specific checkpoint ID are inserted at each source task and broadcasted across all outgoing connections. When an operator received a checkpoint barrier with the same checkpoint ID from all incoming connections, it draws a snapshot of its current state, starts to write it out asynchronously, and broadcasts the checkpoint barrier to all outgoing connections. A checkpoint is completed once all operators finished persisting their state. An important property of checkpoints is that they are consistent, i.e., the snapshotted state of all operators depends on the same set of input records, i.e., all records that were ingested before the checkpoint barriers were injected by the source tasks. In case of a failure, Flink restores the state of all operators from the most recent completed checkpoint. Many source operators persist the read positions (or offsets) on their input streams as regular state, such that read positions and operator states are consistently restored in case of a failure which results in exactly-once state consistency.

Usually, checkpoints are continuously replaced by newer ones. Flink allows for creating

special persistent checkpoints, called *savepoints*. Savepoints can be created at any time and hold the complete state of an application. An application can be restarted from a savepoint, which means its internal state, usually including read positions on input streams, is completely restored. Savepoints make it possible to save multiple versions of a streaming application, recover state, or perform A/B testing with modified business logic but same state in a test environment.

State management also affects whether and how the parallelism of stateful operators can be changed, a feature that is important to support scale out of streaming applications during peak times or increasing loads and scale in afterward to save resources and money. Adjusting the parallelism of a stateful operator requires to redistribute its state to fewer or more parallel tasks. Flink provides mechanisms for scaling operators and applications by taking a savepoint from a running job and restarting it with a different parallelism from the savepoint.

Flink distinguishes between two types of state, *operator state* and *keyed state*. Operator state maintains state that is scoped to the parallel task of an operator. Operator state needs to be splittable and mergeable to support operator rescaling. Keyed state is scoped to a key of the data and requires a keyed stream, i.e., a stream partitioned on a key attribute. When processing a record, an operator with keyed state can only access the state that corresponds to the key of the current record. Operators with keyed state are rescaled by reassigning key ranges and redistributing the corresponding state to fewer or more operator tasks.

Flink maintains the state of operators in a so-called state backend. A state backend implements the internal data structures to hold the data and logic to checkpoint the state. Heap-based state backends use regular Java data structures to store the state. They are limited by the size of the JVM heap of the TaskManagers. Heap-based state backends can checkpoint to the JVM heap of the JobManager or to a remotely accessible file system, such as HDFS, NFS, or S3. The RocksDB-based state backend writes state to disk and therefore allows for state that exceeds the size of available memory. Flink exploits

many RocksDB features for checkpointing to a remote file system, and maintaining state that becomes very large (on the order of multiple terabytes). When using the RocksDB state backend, Flink can not only create checkpoints asynchronously but also incrementally which speeds up the checkpointing operation.

Key Applications

In the past, analytical data was commonly dumped into database systems or distributed file systems and, if at all, processed in nightly or monthly batch jobs. Nowadays, in a globalized, faster-moving world, reacting to events quickly is crucial for economic success. For Flink, processing of unbounded data means handling events when they occur in a non-approximated but accurate fashion. This requires support for event-time processing, exactly-once state consistency, and fault tolerance capabilities. Applications must be scalable to handle both current and increasing data volumes in the future, leading to a parallelized and distributed execution. Moreover, Flink applications can consume data from a variety of sources such as publish-subscribe message queues, file systems, databases, and sockets.

However, the use cases for Flink are not limited to faster data analytics. Its flexibility to model arbitrary dataflows and its precise control over state and time allow a variety of new applications. So-called event-driven applications are becoming more and more popular as part of an overall *event-driven architecture*. Such applications trigger computations based on incoming events and might store an accumulated history of events to relate it to newly arrived events. Applications can output their result to a database or key-value store, trigger an immediate reaction via an RPC call, or emit a new event which might trigger subsequent actions. In contrast to traditional two-tier architectures where application and database system separate business logic and data from each other, event-driven applications own computation and state and keep them close together. The benefits of

this approach are that (1) state access is always local instead of accessing a remote database, (2) state and computation are scaled together, and (3) applications define the schema of their own state, similar to microservices.

Entire social networks have been built with Flink (Koliopoulos 2017) following the event-driven approach. Incoming user requests are written to a distributed log that acts as the single source of truth. The log is consumed by multiple distributed stateful applications which build up their state independently from each other for aggregated post views, like counts, and statistics. Finally, the updated results are stored in a materialized view that web servers can return to the user.

In addition, Flink offers a domain-specific API for complex event processing to detect and react on user-defined patterns in event streams. CEP is useful for fraud or intrusion detection scenarios or to monitor and validate business processes. Furthermore, Flink provides relational APIs to unify queries on bounded and unbounded data streams. With its Table and SQL APIs, Flink can continuously update a result table which is defined by query on one or more input streams, similar to a materialized view as known from relational database systems.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Apache Apex](#)
- ▶ [Apache Samza](#)
- ▶ [Continuous Queries](#)
- ▶ [Definition of Data Streams](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Stream Window Aggregation Semantics and Optimization](#)
- ▶ [Streaming Microservices](#)

References

Akidau T et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in

- massive-scale, unbounded, out-of-order data processing. Proc VLDB Endowment 8(12):1792–1803
- Alexandrov A, Ewen S, Heimel M, Hueske F, Kao O, Markl V, ..., Warneke D (2011) MapReduce and PACT-comparing data parallel programming models. In BTW, pp 25–44
- Alexandrov A, Bergmann R, Ewen S, Freytag JC, Hueske F, Heise A, ..., Naumann F (2014) The stratosphere platform for big data analytics. VLDB J 23(6):939–964
- Batr   D, Ewen S, Hueske F, Kao O, Markl V, Warneke D (2010) Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In: Proceedings of the 1st ACM symposium on cloud computing. ACM, pp 119–130
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015a) Apache Flink: stream and batch processing in a single engine. In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol 36, no. 4
- Carbone P et al (2015b) Apache Flink: stream and batch processing in a single engine. In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol 36, no. 4
- Carbone P et al (2015c) Lightweight asynchronous snapshots for distributed dataflows. In CoRR abs/1506.08603. <http://arxiv.org/abs/1506.08603>
- Carbone P, Ewen S, F  ra G, Haridi S, Richter S, Tzoumas K (2017) State management in apache flink[®]: consistent stateful distributed stream processing. Proc VLDB Endowment 10(12):1718–1729
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
- Ewen S, Tzoumas K, Kaufmann M, Markl V (2012) Spinning fast iterative data flows. Proc VLDB Endowment 5(11):1268–1279
- Ghemawat S, Gobioff H, Leung ST (2003) The google file system. ACM SIGOPS Oper Syst Rev 37(5):29–43. ACM
- Hueske F, Peters M, Sax MJ, Rheinl  nder A, Bergmann R, Krettek A, Tzoumas K (2012) Opening the black boxes in data flow optimization. Proc VLDB Endowment 5(11):1256–1267
- Koliopoulos A (2017) Drivetribe's modern take on CQRS with Apache Flink. Drivetribe. <https://data-artisans.com/blog/drivetribe-cqrs-apache-flink>. Visited on 7 Sept 2017
- Mani Chandy K, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM Trans Comp Syst (TOCS) 3(1):63–75
- The Apache Software Foundation. RocksDB|A persistent key-value store|RocksDB. <http://rocksdb.org/>. Visited on 30 Sept 2017
- Hueske F, Kalavri V (2018) Stream processing with Apache Flink: fundamentals, implementation, and operation of streaming applications. O'Reilly Media, Sebastopol. ISBN 149197429X

Apache Hadoop

► Architectures

Apache Kafka

Matthias J. Sax
Confluent Inc., Palo Alto, CA, USA

Definitions

Apache Kafka (Apache Software Foundation 2017b; Kreps et al. 2011; Goodhope et al. 2012; Wang et al. 2015; Kleppmann and Kreps 2015) is a scalable, fault-tolerant, and highly available distributed streaming platform that can be used to store and process data streams.

Kafka consists of three main components:

- the Kafka cluster,
- the Connect framework (Connect API),
- and the Streams programming library (Streams API).

The Kafka cluster stores data streams, which are sequences of messages/events continuously produced by applications and sequentially and incrementally consumed by other applications. The Connect API is used to ingest data into Kafka and export data streams to external systems like distributed file systems, databases, and others. For data stream processing, the Streams API allows developers to specify sophisticated stream processing pipelines that read input streams from the Kafka cluster and write results back to Kafka.

Kafka supports many different use cases categories such as traditional publish-subscribe

Recommended Reading

- Friedman E, Tzoumas K (2016) Introduction to Apache Flink: stream processing for real time and beyond. O'Reilly Media, Sebastopol. ISBN 1491976586

messaging, streaming ETL, and data stream processing.

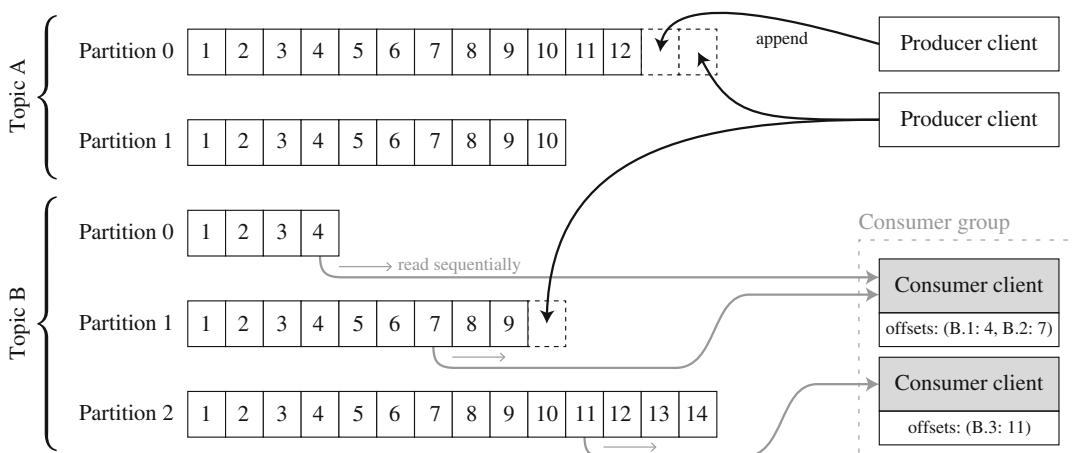
Overview

A Kafka cluster provides a publish-subscribe messaging service (Fig. 1). Producer clients (publishers) write messages into Kafka, and consumer clients (subscribers) read those messages from Kafka. *Messages* are stored in Kafka servers called *brokers* and organized in named *topics*. A topic is an append-only sequence of messages, also called a log. Thus, each time a message is written to a topic, it is appended to the end of the log. A Kafka message is a key-value pair where key and value are variable-length byte arrays. Additionally, each message has a time stamp that is stored as 64-bit integer.

Topics are divided into *partitions*. When a message is written to a topic, the producer must specify the partition for the message. Producers can use any partitioning strategy for the messages they write. By default, messages are hash-partitioned by key, and thus all messages with the same key are written to the same partition. Each message has an associated *offset* that is the message's position within the partition, i.e., a monotonically increasing sequence number. The mes-

sage's offset is implicitly determined by the order in which messages are appended to a partition. Hence, each message within a topic is uniquely identified by its partition and offset. Kafka guarantees strict message ordering within a single partition, i.e., it guarantees that all consumers reading a partition receive all messages in the exact same order as they were appended to the partition. There is no ordering guarantee between messages in different partitions or different topics.

Topics can be written by multiple producers at the same time. If multiple producers write to the same partition, their messages are interleaved. If consumers read from the same topic, they can form a so-called consumer group. Within a consumer group, each individual consumer reads data from a subset of partitions. Kafka ensures that each partition is assigned to exactly one consumer within a group. Different consumer groups or consumers that don't belong to any consumer group are independent from each other. Thus, if two consumer groups read the same topic, all messages are delivered to both groups. Because consumers are independent from each other, each consumer can read messages at its own pace. This results in decoupling—a desirable property for a distributed system—and makes the system robust against stragglers. In summary, Kafka supports



Apache Kafka, Fig. 1 Kafka topics are divided into partitions that are ordered sequences of messages. Multiple producers can write simultaneously into the same topic. Consumers track their read progress and can form

a consumer group to share the read workload over all consumers within the group (Source: Kleppmann and Kreps 2015)

multiple producers and can deliver the same data to multiple consumers.

Kafka Brokers

Kafka brokers store messages reliably on disk. In contrast to traditional messaging/publish-subscribe systems, Kafka can be used for long-term storage of messages, because Kafka does not delete messages after delivery. Topics are configured with a so-called *retention time* that specifies how long a message should be stored. Topic retention can also be specified in bytes instead of time, to apply an upper bound on disk space. If the retention boundaries are reached, Kafka truncates partitions at the end of the log.

From a semantic point of view, messages are immutable facts, and thus it is not reasonable to support deleting individual messages from a topic. Users can only apply a “time-to-live” via topic retention to truncate old data.

Log Compaction

Kafka also supports so-called *compacted* topics. If a topic is configured for log compaction, users apply different *semantics* to the stored messages. While regular topics store *immutable facts*, a compacted topic can be used to store *updates*. Note that a compacted topic is still an append-only sequence of messages, and there are no in-place updates. Appending an update message to a compacted topic implies that a newer messages “replaces” older messages with the same key. The difference of compacted topics to topics with log retention is that Kafka guarantees that the latest update of a key is never deleted, while older updates can be garbage collected.

Log compaction is applied on a per-partition basis; thus updates for the same key should be written to the same partition. For performance reasons, brokers don’t delete older messages immediately, but compaction is triggered as background process in regular intervals. In contrast to “regular” topics, compacted topics also support delete semantics for individual record via so-called *tombstone* messages. A tombstone is a

message with a null value, and it indicates that all previous updates for the corresponding key can be deleted (including the tombstone itself).

Scaling and Load Balancing

As described in section “[Overview](#),” messages are stored in topics, and topics are divided into partitions. Partitions allow brokers to scale out horizontally and to balance load within the cluster, because partitions are independent units within a topic. Even if partitions of the same topic are stored at different brokers, there is no need for broker synchronization, and thus a Kafka cluster scales linearly with the number of brokers. A single broker only limits the capacity of a single partition, but because topics can be created with an arbitrary number of partitions, this is not a limitation in practice. Overall the read/write throughput and storage requirements of topics are not limited by the size of a single server, and the cluster capacity can be increased by adding new brokers to the system. Last but not least, partitions can be reassigned from one broker to another to balance load within a cluster.

There is no master node in a Kafka cluster: all brokers are able to perform all services provided by the cluster. This design supports linear scale-out, as a master node could become a bottleneck. For broker coordination, Kafka uses Apache ZooKeeper (Apache Software Foundation [2017d](#); Hunt et al. [2010](#)), a scalable, fault-tolerant, and highly available distributed coordination service. Kafka uses ZooKeeper to store all cluster metadata about topics, partitions, partition-to-broker mapping, etc., in a reliable and highly available manner.

Fault Tolerance and High Availability

To ensure fault tolerance and high availability, partitions can be replicated to multiple brokers. Each topic can be configured with an individual replication factor that indicates how many copies of a partition should be maintained. To ensure strict message ordering guarantees per partitions, replication uses a *leader-follower* pattern. Each partition has a single leader and a configurable number of followers. All read and write requests are handled by the leader, while the followers

replicate all writes to the leader in the background. If the broker hosting the leader fails, Kafka initiates a leader election via ZooKeeper, and one of the followers becomes the new leader. All clients will be updated with the new leader information and send all their read/write request to the new leader. If the failed broker recovers, it will rejoin the cluster, and all hosted partitions become followers.

Message Delivery

Reading data from Kafka works somewhat differently compared to traditional messaging/publish-subscribe systems. As mentioned in section “[Kafka Brokers](#),” brokers do not delete messages after delivery. This design decision has multiple advantages:

- Brokers do not need to track the reading progress of consumers. This allows for an increased read throughput, as there is no progress tracking overhead for the brokers.
- It allows for in-order message delivery. If brokers track read progress, consumers need to acknowledge which messages they have processed successfully. This is usually done on a per-message basis. Thus, if an earlier message is not processed successfully, but a later message is processed successfully, re-delivery of the first message happens out of order.
- Because brokers don’t track progress and don’t delete data after delivery, consumers can go back in time and reprocess old data again. Also, newly created consumers can retrieve older data.

The disadvantage of this approach is that consumer clients need to track their progress themselves. This happens by storing the offset of the next message a consumer wants to read. This offset is included in the read request to the broker, and the broker will deliver consecutive messages starting at the requested offset to the consumer. After processing all received messages, the consumer updates its offset accordingly and sends the next read request to the cluster.

If a consumer is stopped and restarted later on, it usually should continue reading where it left off. To this end, the consumer is responsible for storing its offset reliably so it can retrieve it on restart. In Kafka, consumers can commit their offsets to the brokers. On offset commit brokers store consumer offsets reliably in a special topic called *offset topic*. As offset topic is also partitioned and replicated and is thus scalable, fault-tolerant, and highly available. This allows Kafka to manage a large number of consumers at the same time. The offset topic is configured with log compaction enabled (cf. section “[Log Compaction](#)”) to guarantee that offsets are never lost.

Delivery Semantics

Kafka supports multiple delivery semantics, namely, *at-most-once*, *at-least-once*, and *exactly once*. What semantics a user gets depends on multiple factors like cluster/topic configuration as well as client configuration and user code.

It is important to distinguish between the write and read path when discussing delivery semantics. Furthermore, there is the concept of end-to-end processing semantics that applies to Kafka’s Streams API. In this section, we will only cover the read and write path and refer to section “[Kafka Streams](#)” for end-to-end processing semantics.

Writing to Kafka

When a producer writes data into a topic, brokers acknowledge a successful write to the producer. If a producer doesn’t receive an acknowledgement, it can ignore this and follow at-most-once semantics, as the message might not have been written to the topic and thus could be lost. Alternatively, a producer can retry the write resulting in at-least-once semantics. The first write could have been successful, but the acknowledgement might be lost. Kafka also supports exactly once writes by exploiting idempotence. For this, each message is internally assigned a unique identifier. The producer attaches this identifier to each message it writes, and the broker stores the identifier as metadata of each message in the topic. In case of a producer write retry, the broker can detect the

duplicate write by comparing the message identifiers. This deduplication mechanism is internal to producer and broker and does not guard against application-level duplicates.

Atomic Multi-partition Writes Kafka also support atomic multi-partition writes that are called *transactions*. A Kafka transaction is different to a database transaction, and there is no notion of ACID guarantees. A transaction in Kafka is similar to an atomic write of multiple messages that can span different topics and/or partitions. Due to space limitations, we cannot cover the details of transactional writes and can only give a brief overview. A transactional producer is first initialized for transactions by performing a corresponding API call. The broker is now ready to accept transactional writes for this producer. All messages sent by the producer belong to the current transaction and won't be delivered to any consumer as long as the transaction is not completed. Messages within a transaction can be sent to any topic/partition within the cluster. When all messages of a transaction have been sent, the producer commits the transaction. The broker implements a two-phase commit protocol to commit a transaction, and it either successfully "writes" all or none of the messages belonging to a transaction. A producer can also abort a transaction; in this case, none of the messages will be deleted from the log, but all messages will be marked as aborted.

Reading from Kafka

As discussed in section "[Message Delivery](#)," consumers need to track their read progress themselves. For fault-tolerance reasons, consumers commit their offsets regularly to Kafka. Consumers can apply two strategies for this: after receiving a message, they can either first commit the offset and process the message afterwards, or they do it in reverse order and first process the message and commit the offset at the end. The commit-first strategy provides at-most-once semantics. If a message is received and the offset is committed before the message is processed, this message would not be redelivered in case of failure: after a failure, the consumer would

recover its offsets as the last committed offset and thus would resume reading after the failed message.

In contrast, the process-first strategy provides at-least-once semantics. Because the consumer doesn't update its offsets after processing the receive message successfully, it would always fall back to the old offset after recovering from an error. Thus, it would reread the processed message, resulting in potential duplicates in the output.

Transactional Consumers Consumers can be configured to read all (including aborted messages) or only committed messages (cf. paragraph *Atomic multi-partition writes* in section "[Writing to Kafka](#)"). This corresponds to a read uncommitted and read committed mode similar to other transactional systems. Note that aborted messages are not deleted from the topics and will be delivered to all consumers. Thus, consumers in read committed mode will filter/drop aborted messages and not deliver them to the application.

Kafka Connect Framework

Kafka Connect—or the Connect API—is a framework for integrating Kafka with external systems like distributed file systems, databases, key-value stores, and others. Internally, Kafka Connect uses producer/consumer clients as described in section "[Kafka Brokers](#)," but the framework implements much of the functionality and best practices that would otherwise have to be implemented for each system. It also allows running in a fully managed, fault-tolerant, and highly available manner.

The Connect API uses a *connector* to communicate with each type of external system. A *source connector* continuously reads data from an external source system and writes the records into Kafka, while a *sink connector* continuously consumes data from Kafka and sends the records to the external system. Many connectors are available for a wide variety of systems, including HDFS, S3, relational databases, document database systems, other messaging systems, file

systems, metric systems, analytic systems, and so on. Developers can create connectors for other systems.

Kafka Connect can be either used as stand-alone or deployed to a cluster of machines. To connect with an external system, a user creates a configuration for a connector that defines the specifics of the external system and the desired behavior, and the user uploads this configuration to one of the Connect workers. The worker, which is running in a JVM, deploys the connector and distributes the connector's *tasks* across the cluster. Source connector tasks load data from the external system and generate records, which Connect then writes to the corresponding Kafka topics. For sink connector tasks, Connect consumes the specified topics and passes these records to the task, which then is responsible for sending the records to the external system.

Single-Message Transforms

The Connect API allows to specify simple transformation—so-called single-message transforms, SMT—for individual messages that are imported/exported into/from Kafka. Those transformations are independent of the connector and allow for stateless operations. Standard transformation functions are already provided by Kafka, but it is also possible to implement custom transformations to perform initial data cleaning. If SMTs are not sufficient because a more complex transformation is required, the Streams API (described in the next section) can be used instead.

Kafka Streams

Kafka Streams—or the Streams API—is the stream processing library of Apache Kafka. It provides a high-level DSL that supports stateless as well as stateful stream processing operators like joins, aggregations, and windowing. The Streams API also supports exactly once processing guarantees and event-time semantics and handles out-of-order and late-arriving data. Additionally, the Streams API introduces *tables*

as a first-class abstraction next to data *streams*. It shares a few ideas with Apache Samza (cf. article on “► [Apache Samza](#)”) (Apache Software Foundation 2017c; Noghabi et al. 2017; Kleppmann and Kreps 2015) such as building on top of Kafka's primitives for fault tolerance and scaling. However, there are many notable differences to Samza: for example, Kafka Streams is implemented as a library and can run in any environment, unlike Samza, which is coupled to Apache Hadoop's resource manager YARN (Apache Software Foundation 2017a; Vavilapalli et al. 2013); it also provides stronger processing guarantees and supports both streams and tables as core data abstractions.

Streams and Tables

Most stream processing frameworks provide the abstraction of a *record stream* that is an append-only sequence of immutable facts. Kafka Streams also introduces the notion of a *changelog stream*, a mutable collection of data items. A changelog stream can also be described as a continuously updating table. The analogy between a changelog and a table is called the *stream-table duality* (Kleppmann 2016, 2017). Supporting tables as first-class citizens allow to enrich data streams via stream-table joints or populate tables as self-updating caches for an application.

The concept of changelogs aligns with the concept of compacted topics (cf. section “[Log Compaction](#)”). A changelog can be stored in a compacted topic, and the corresponding table can be recreated by reading the compacted topic without data loss as guaranteed by the compaction contract.

Kafka Streams also uses the idea of a changelog stream for (windowed) aggregations. An aggregation of a record stream yields both a table and a changelog stream as a result—not a record stream. Thus, the table always contains the current aggregation result that is updated for each incoming record. Furthermore, each update to the result table is propagated downstream as a changelog record that is appended to the changelog stream.

State Management

Operator state is a first-class citizen in Kafka's Streams API similar to Samza and uses the aforementioned table abstraction. For high performance, state is kept local to the stream processing operators using a RocksDB (Facebook Inc. 2017) store. Local state is not fault-tolerant, and thus state is additionally backed by a topic in the Kafka cluster. Those topics have log compaction (cf. section “[Log Compaction](#)”) enabled and are called *changelog topics*. Using log compaction ensures that the size of the changelog topic is linear in the size of the state. Each update to the store is written to the changelog topic; thus, the persistent changelog topic is the source of truth, while the local RocksDB store is an ephemeral materialized view of the state.

If an application instance fails, another instance can recreate the state by reading the changelog topic. For fast fail-over, Kafka Streams also support *standby replicas*, the hold hot standbys of state store. Standby replicas can be maintained by continuously reading all changes to the primary store from the underlying changelog topic.

Fault Tolerance and Scaling

Kafka Streams uses the same scaling/parallelism abstraction as Samza, namely, partitions. This is a natural choice as Kafka Streams reads input data from partitioned topics. Each input topic partition is mapped to a *task* that processes the records of this partition. Tasks are independent units of parallelism and thus can be executed by different threads that might run on different machines. This allows to scale out a Kafka Streams application by starting multiple instances on different machines. All application instances form a consumer group, and thus Kafka assigns topic partitions in a load balanced manner to all application instances (cf. section “[Kafka Brokers](#)”).

Because Kafka Streams is a library, it cannot rely on automatic application restarts of failed instances. Hence, it relies on the Kafka cluster to detect failures. As mentioned above, a Streams application forms a consumer group, and the Kafka cluster monitors the liveness of

all members of the group. In case of a failure, the cluster detects a dead group member and reassigns the corresponding input topic partitions of the failed application instance to the remaining instances. This process is called an consumer group *rebalance*. During a rebalance, Kafka Streams also ensures that operator state is migrated from the failing instance (cf. section “[State Management](#)”). Kafka's consumer group management mechanism also allows for fully elastic deployments. Application instances can be added or removed during runtime without any downtime: the cluster detects joining/leaving instances and rebalances the consumer group automatically. If new instances are joining, partitions are *revoked* from existing members of the groups and assigned to the new members to achieve load balancing. If members are leaving a consumer group, this is just a special case of fail-over, and the partitions of the leaving instance are assigned to the remaining ones. Note that in contrast to a fail-over rebalance, a scaling rebalance guarantees a clean hand over of partitions, and thus each record is processed exactly once.

Time Semantics

Time is a core concept in stream processing, and the operators in Kafka's stream processing DSL are based on time: for example, windowed aggregations require record time stamps to assign records to the correct time windows. Handling time also requires to handle late-arriving data, as record might be written to a topic out of order (note: Kafka guarantees offset based in-order delivery; there is no time stamp-based delivery guarantee).

Stream processing with Kafka supports three different time semantics: event time, ingestion time, and processing time. From a Streams API point of view, there are only two different semantics though: *event time* and *processing time*.

Processing time semantics are provided when there is no record time stamp, and thus Streams needs to use wall clock time when processing data for any time-based operation like windowing. Processing time has the disadvantage that there is no relationship between the time when data was created and when data gets processed.

Furthermore, processing time semantics is inherently non-deterministic.

Event time semantics are provided when the record contains a time stamp in its payload or metadata. This time-stamp is assigned when a record is created and thus allows for deterministic data reprocessing. Applications may include a time stamp in the payload of the message, but the processing of such time stamps is then application-dependent. For this reason, Kafka supports record metadata time stamps that are stored in topics and automatically set by the producer when a new record is created.

Additionally, Kafka topics can be configured to support *ingestion time* for time-based operations: ingestion time is the time when data is appended to the topic, i.e., the current broker wall clock time on write. Ingestion time is an approximation of event time, assuming that data is written to a topic shortly after it was created. It can be used if producer applications don't provide a record metadata time stamp. As ingestion time is an approximation of event time, the Streams API is agnostic to ingestion time (it is treated as a special case of event time).

Exactly Once Processing Semantics

Processing an input record can be divided into three parts: first, the actual processing including any state updates; second, writing the result records to the output topics; third, recording the progress by committing the consumer offset. To provide exactly once processing semantics, all three parts must be performed “all or nothing.”

As described in section “[State Management](#),” updating operator state is actually a write to a changelog topic that is the source of truth (the local state is only a materialized view). Furthermore, committing input offsets is a write to the special *offset topic* as discussed in section “[Message Delivery](#).“ Hence, all three parts use the same underlying operation: writing to a topic.

This allows the Streams API to leverage Kafka's transactions (cf. section “[Writing to Kafka](#)”) to provide end-to-end exactly once stream processing semantics. All writes that happen during the processing of a record are part of the same transaction. As a transaction is

an atomic multi-partition write, it ensures that either all writes are committed or all changes are aborted together.

Summary

Apache Kafka is a scalable, fault-tolerant, and highly available distributed streaming platform. It allows fact and changelog streams to be stored and processed, and it exploits the stream-table duality in stream processing. Kafka's transactions allow for exactly once stream processing semantics and simplify exactly once end-to-end data pipelines. Furthermore, Kafka can be connected to other systems via its Connect API and can thus be used as the central data hub in an organization.

Cross-References

- ▶ [Apache Flink](#)
- ▶ [Apache Samza](#)
- ▶ [Continuous Queries](#)

References

- Apache Software Foundation (2017a) Apache Hadoop project web page. <https://hadoop.apache.org/>
- Apache Software Foundation (2017b) Apache Kafka project web page. <https://kafka.apache.org/>
- Apache Software Foundation (2017c) Apache Samza project web page. <https://samza.apache.org/>
- Apache Software Foundation (2017d) Apache ZooKeeper project web page. <https://zookeeper.apache.org/>
- Facebook Inc (2017) RocksDB project web page. <http://rocksdb.org/>
- Goodhope K, Koshy J, Kreps J, Narkhede N, Park R, Rao J, Ye VY (2012) Building LinkedIn's real-time activity data pipeline. IEEE Data Eng Bull 35(2):33–45. <http://sites.computer.org/debull/A12june/pipeline.pdf>
- Hunt P, Konar M, Junqueira FP, Reed B (2010) ZooKeeper: wait-free coordination for internet-scale systems. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIX ATC'10. USENIX Association, Berkeley, p 11. <http://dl.acm.org/citation.cfm?id=1855840.1855851>
- Kleppmann M (2016) Making sense of stream processing, 1st edn. O'Reilly Media Inc., 183 pages

- Kleppmann M (2017) Designing data-intensive applications. O'Reilly Media Inc., Sebastopol
- Kleppmann M, Kreps J (2015) Kafka, Samza and the Unix philosophy of distributed data. IEEE Data Eng Bull 38(4):4–14. <http://sites.computer.org/debull/A15dec/p4.pdf>
- Kreps J, Narkhede N, Rao J (2011) Kafka: a distributed messaging system for log processing. In: Proceedings of the NetDB, pp 1–7
- Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza: stateful scalable stream processing at LinkedIn. Proc VLDB Endow 10(12):1634–1645. <https://doi.org/10.14778/3137765.3137770>
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E (2013) Apache Hadoop YARN: yet another resource negotiator. In: 4th ACM symposium on cloud computing (SoCC). <https://doi.org/10.1145/2523616.2523633>
- Wang G, Koshy J, Subramanian S, Paramasivam K, Zadeh M, Narkhede N, Rao J, Kreps J, Stein J (2015) Building a replicated logging system with Apache Kafka. PVLDB 8(12):1654–1655. <http://www.vldb.org/pvldb/vol8/p1654-wang.pdf>

Apache Mahout

Andrew Musselman
Apache Software Foundation, Seattle, WA, USA

Definitions

Apache Mahout (<http://mahout.apache.org>) is a distributed linear algebra framework that includes a mathematically expressive domain-specific language (DSL). It is designed to aid mathematicians, statisticians, and data scientists to quickly implement numerical algorithms while focusing on the mathematical concepts in their work, rather than on code syntax. Mahout uses an extensible plug-in interface to systems such as Apache Spark and Apache Flink.

Historical Background

Mahout was founded as a sub-project of Apache Lucene in late 2007 and was promoted to a top-

level Apache Software Foundation (ASF) (ASF 2017) project in 2010 (Khudairi 2010). The goal of the project from the outset has been to provide a machine learning framework that was both accessible to practitioners and able to perform sophisticated numerical computation on large data sets.

Mahout has undergone two major stages of architecture design. The first versions relied on the Apache Hadoop MapReduce framework, a popular tool for orchestrating large-scale data flow and computation. Hadoop, while flexible enough to handle many typical workflows, has severe limitations when applied to highly iterative processes, like those used in most machine learning methods.

The Hadoop implementation of MapReduce does not allow for caching of intermediate results across steps of a long computation. This means that algorithms that iteratively use the same data many times are at a severe disadvantage – the framework must read data from disk every iteration rather than hold it in memory. This type of algorithm is common in data science and machine learning: examples include k -means clustering (which has to compute distances for all points in each iteration) and stochastic gradient descent (which has to compute gradients for the same points over many iterations).

Since enabling iterative work on large data sets is a core requirement of a machine learning library geared toward big data, Mahout moved away from Hadoop in its second design phase. Starting with release 0.10.0 (PMC 2015), Mahout switched its computation engine to Apache Spark, a framework designed specifically to facilitate distributed in-memory computation. At the same time, Mahout jobs built using Hadoop MapReduce were deprecated to discourage users from relying on them long-term, and a new interface on the front end was released, named “Samsara,” after the Hindu and Buddhist concept of rebirth. This new front end has a simple syntax for matrix math modeled after other systems such as MATLAB and R, to allow maximal readability and quick prototyping. Samsara can be run in an interactive shell where results are displayed inline, enabling a live back-and-forth with code and

results, much like an interactive SQL or other interpreted environment familiar to many users.

As an example of the Samsara DSL syntax for calculating, for instance, the transpose of a matrix

A multiplied with A itself, a common operation in machine learning jobs, can be written in Scala code as

```
val C = A.t %*% A.
```

Note the declaration of the resulting matrix C , the transpose operator t , and the matrix multiplication operator $\%*\%$. Behind the scenes, the right-hand side of the statement is parsed and the back end takes an optimal approach when physically performing the multiplication.

More recently, beginning in early 2017 with version 0.13.0 (PMC 2017), Mahout added the capability to perform computation directly on hardware, outside the Java Virtual Machine (JVM), using operation solvers written specifically to optimize performance according to native processor and memory architecture. This allows for speed gains from using all cores of all CPUs on the host and the option to leverage matrix-math-specific instruction sets built into graphics processing units (GPUs).

Foundations

The motivation for the Mahout project is to bring large-scale machine learning to practitioners in real-world environments. This boils down to three concepts: first, providing a convenient syntax for writing programs and doing experimental and exploratory work; second, handling all the nitty-gritty details of distributed matrix arithmetic without requiring the user to understand the low-level implementation; and last, making deployment to production simple and straightforward.

Many machine learning libraries and tools either provide an array of algorithms, but do not operate on very large data sets, or else manage computation at scale but require significant programming experience and skill. Bridging those gaps, Mahout works at scale, is flexible with

regard to hardware and compute engines, and allows data science and machine learning practitioners to focus on the math in their work symbolically.

For example, in the distributed stochastic principal component analysis (dSPCA) job in Mahout, there is a computation that is expressed symbolically as

$$G = BB^T - C - C^T + \xi^T \xi s_q^T s_q.$$

This equation can be expressed in code in Mahout as

```
val G = B %*% B.t - C - C.t
+ (xi dot xi) * (s_q cross s_q),
```

which is readable and compact compared to many alternative representations.

The main reason many machine learning operations need to be distributed in real-world environments is because one of the key strengths of machine learning methods is their ability to build predictions from very large data sets. Data sets at this scale include, for example, HTTP access logs for high-traffic websites, large text corpora consisting of millions of documents, and streaming, high-volume data from sensors in industrial settings. With smaller data sets, the input is often held in memory, and all computation is performed on one host computer. However, as soon as a data set scales beyond the memory on a single host, inefficiency caused by spilling data to storage and re-reading it slows down operations and makes them too costly to be viable.

Distributed computing frameworks that allow computation on large data sets are continuously evolving based on shifting market requirements and ongoing research and development. Mahout takes advantage of many of the abstractions provided by these frameworks and then builds further performance improvements and refinements. Many refinements come from exploiting well-studied properties of matrices and matrix arithmetic and using “tricks” and shortcuts to avoid doing more operations than required,

and others come from solvers purpose-built for specific types of hardware and environments.

Overview of Architecture

The most recent design of Mahout allows for iterative rapid prototyping in the Samsara DSL which can be used directly in production with minimal changes. Computation on the back end is adaptable to varied configurations of compute engine and hardware. For example, an algorithm can be handed off to a Spark or Flink engine for computation, and any required matrix arithmetic can be computed in the JVM, on the CPU (using all available cores), or on compatible GPUs, depending on the shape and characteristics of the vectors and matrices being operated on. All these combinations will work with the same front-end code with minimal changes.

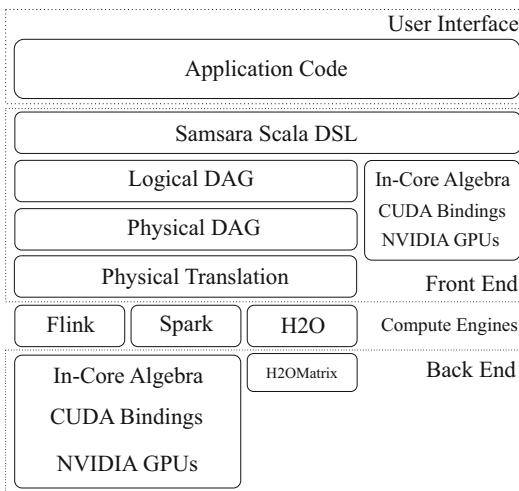
All the moving parts involved are seen in Fig. 1, moving from the top layer where application code is written to the front end which hands off computation to the appropriate engine or to native solvers (which perform the com-

putations), the pluggable compute engine layer, and the back end which handles computation according to instructions from compute engines. Noteworthy for the Mahout user is the Scala DSL layer which allows for interactive, iterative development which can be captured directly into Scala packages and run as application code in production environments with minimal changes.

Also notable is the pluggable compute engine layer, allowing flexibility in deployment as well as future-proofing for when new compute engines become viable and beneficial to users' needs. In fact, code written for the Spark engine, for example, can be directly re-used on Flink with minor changes to import and initialization statements in the code.

Another recent introduction to Mahout is a template that simplifies algorithm contributions from the project's maintainers as well as from its users. The interface is modeled after the machine learning training and evaluation patterns found in many R packages and the Python-based scikit-learn package.

As seen in Fig. 2, a new algorithm need only define a `fit` method in a `Fitter` class, which populates a `Model` class, which contains parameter estimates, statistics about the fit, and an overall summary of performance, and which finally uses its `predict` method to make predictions on new data. The bulk of the work any new algorithm performs is written inside the `fit` method.



Apache Mahout, Fig. 1 Current architecture of Mahout, showing four main components: the user interface where application code is written, the front end which handles which subsystem will perform computation (which could be native and in-core), the pluggable compute engine layer, and the back end which takes instructions from a compute engine and performs mathematical operations

Key Applications

Practical applications of machine learning usually fall into one of three categories, the so-called Three Cs: collaborative filtering (known commonly as “recommender systems,” or “recommenders”), classification, and clustering.

Recommender systems typically come into play in situations where the goal is to present new items to users which they are likely to need or want. These recommendations are based on historical user behavior across all users and across all items they interact with. Examples

Apache Mahout, Fig. 2

Code listing of a skeleton for a new algorithm, showing both the Fitter and Model classes, including their fit and predict methods

```

class Foo[K] extends RegressorFitter[K] {

    def fit(drmX: DrmLike[K],
            drmTarget: DrmLike[K],
            hyperparameters: (Symbol, Any)*): FooModel[K] = {
        /**
         * Normally this section would have more code
         *
         */
        var model = new FooModel[K]
        model.summary = "This model has been fit, etc."
        model
    }
}

class FooModel[K] extends RegressorModel[K] {

    def predict(drmPredictors: DrmLike[K]): DrmLike[K] = {
        drmPredictors.mapBlock(1) {
            case (keys, block: Matrix) => {
                var outputBlock = new DenseMatrix(block.nrow, 1)
                keys -> (outputBlock += 1.0)
            }
        }
    }
}

```

include online retail, where increasing customer spending and site interactions is a key business goal. A common use of a recommender is to present examples of products that other customers have bought, calculated by the similarity between other customers' purchases or between products themselves.

Classifiers cover a broad range of methods which can be summarized as predicting either a categorical value for a user or an item, such as “likely or unlikely to default on a bank loan,” or a numerical value such as age or salary. These methods are used across most scientific fields and industrial sectors, for anything from screening for illnesses in medicine, to determining risk in the financial sector, to predictive plant maintenance in manufacturing operations.

Clustering methods are generally used to make sense of groups of users, documents, or other items. Presented as a whole, a data set can be daunting or even impossible to understand as a pile of records full of numbers and values, and it is often desirable to segment the whole set into smaller pieces, each of which collects most-similar items together for more close analysis and inspection. A common use for clustering is

with a text corpus that contains documents on a variety of subjects. In order to categorize all the documents into one or more relevant topics which could be used for quicker filing and retrieval, the entire corpus can be analyzed by a clustering method which separates documents into groups, members of each being relevant to similar topics or concepts.

In real-world applications, the distinction between these methods can be less clear-cut. Often more than one type of machine learning method will be used in combination to achieve further nuance and improve relevance and effectiveness of predictions. For example, after a corpus of documents is organized into topics, if a new document is added to the corpus, it can effectively be “classified” based on which cluster it is closest to. Similarly, the input to a recommender system in production can be a customer’s recent browsing history on the product website. The customer’s history can be fed into a classifier to determine which cohort of customers they belong to. This information can then be the input to a recommender built specifically per cohort, so that any products recommended are more relevant to that customer.

Cross-References

- ▶ [Apache Flink](#)
- ▶ [Apache Hadoop](#)
- ▶ [Apache Spark](#)
- ▶ [Apache SystemML](#)
- ▶ [Big Data and Recommendation](#)
- ▶ [Columnar Storage Formats](#)
- ▶ [GPU-Based Hardware Platforms](#)
- ▶ [Python](#)
- ▶ [Scala](#)
- ▶ [Scalable Architectures for Big Data Analysis](#)

References

- ASF (2017) Welcome to the apache software foundation! <https://www.apache.org>
- Khudairi S (2010) The apache software foundation blog. https://blogs.apache.org/foundation/entry/the_apache_software_announces4
- PMC AM (2015) Apache mahout 0.10.0 release notes. <http://mahout.apache.org/release-notes/Apache-Mahout-0.10.0-Release-Notes.pdf>
- PMC AM (2017) Apache mahout 0.13.0 release notes. https://mail-archives.apache.org/mod_mbox/www-announce/201704.mbox/%3CCANg8BGBe+WwdZC6z6BAm3hqTOMjA2ma76y0dig0Jf5LHtg_F56g@mail.gmail.com%3E

Apache Samza

Martin Kleppmann
University of Cambridge, Cambridge, UK

Definitions

Apache Samza is an open source framework for distributed processing of high-volume event streams. Its primary design goal is to support high throughput for a wide range of processing patterns, while providing operational robustness at the massive scale required by Internet companies. Samza achieves this goal through a small number of carefully designed abstractions: partitioned

logs for messaging, fault-tolerant local state, and cluster-based task scheduling.

Overview

Stream processing is playing an increasingly important part of the data management needs of many organizations. Event streams can represent many kinds of data, for example, the activity of users on a website, the movement of goods or vehicles, or the writes of records to a database.

Stream processing jobs are long-running processes that continuously consume one or more event streams, invoking some application logic on every event, producing derived output streams, and potentially writing output to databases for subsequent querying. While a batch process or a database query typically reads the state of a dataset at one point in time, and then finishes, a stream processor is never finished: it continually awaits the arrival of new events, and it only shuts down when terminated by an administrator.

Many tasks can be naturally expressed as stream processing jobs, for example:

- aggregating occurrences of events, e.g., counting how many times a particular item has been viewed;
- computing the rate of certain events, e.g., for system diagnostics, reporting, and abuse prevention;
- enriching events with information from a database, e.g., extending user click events with information about the user who performed the action;
- joining related events, e.g., joining an event describing an email that was sent with any events describing the user clicking links in that email;
- updating caches, materialized views, and search indexes, e.g., maintaining an external full-text search index over text in a database;
- using machine learning systems to classify events, e.g., for spam filtering.

Apache Samza, an open source stream processing framework, can be used for any of the above applications (Kleppmann and Kreps 2015; Noghabi et al. 2017). It was originally developed at LinkedIn, then donated to the Apache Software Foundation in 2013, and became a top-level Apache project in 2015. Samza is now used in production at many Internet companies, including LinkedIn (Paramasivam 2016), Netflix (Netflix Technology Blog 2016), Uber (Chen 2016; Hermann and Del Balso 2017), and TripAdvisor (Calisi 2016).

Samza is designed for usage scenarios that require very high throughput: in some production settings, it processes millions of messages per second or trillions of events per day (Feng 2015; Paramasivam 2016; Noghabi et al. 2017). Consequently, the design of Samza prioritizes scalability and operational robustness above most other concerns.

The core of Samza consists of several fairly low-level abstractions, on top of which high-level operators have been built (Pathirage et al. 2016). However, the core abstractions have been carefully designed for operational robustness, and the scalability of Samza is directly attributable to the choice of these foundational abstractions. The remainder of this article provides further detail on

those design decisions and their practical consequences.

Partitioned Log Processing

A Samza job consists of a set of Java Virtual Machine (JVM) instances, called *tasks*, that each processes a subset of the input data. The code running in each JVM comprises the Samza framework and user code that implements the required application-specific functionality. The primary API for user code is the Java interface `StreamTask`, which defines a method `process()`. Figure 1 shows two examples of user classes implementing the `StreamTask` interface.

Once a Samza job is deployed and initialized, the framework calls the `process()` method once for every message in any of the input streams. The execution of this method may have various effects, including querying or updating local state and sending messages to output streams. This model of computation is closely analogous to a map task in the well-known MapReduce programming model (Dean and Ghemawat 2004), with the difference that

```
class SplitWords implements StreamTask {
    static final SystemStream WORD_STREAM =
        new SystemStream("kafka", "words");

    public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {
        String str = (String) in.getMessage();
        for (String word : str.split(" ")) {
            out.send(
                new OutgoingMessageEnvelope(
                    WORD_STREAM, word, 1));
        }
    }
}
```

```
class CountWords implements StreamTask,
    IitableTask {
    private KeyValueStore<String, Integer> store;

    public void init(Config config,
                     TaskContext context) {
        store = (KeyValueStore<String, Integer>)
            context.getStore("word-counts");
    }

    public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {
        String word = (String) in.getKey();
        Integer inc = (Integer) in.getMessage();
        Integer count = store.get(word);
        if (count == null) count = 0;
        store.put(word, count + inc);
    }
}
```

Apache Samza, Fig. 1 The two operators of a streaming word-frequency counter using Samza's `StreamTask` API (Image source: Kleppmann and Kreps 2015, © 2015 IEEE, reused with permission)

a Samza job’s input is typically never-ending (*unbounded*).

Similarly to MapReduce, each Samza task is a single-threaded process that iterates over a sequence of input records. The inputs to a Samza job are partitioned into disjoint subsets, and each input partition is assigned to exactly one processing task. More than one partition may be assigned to the same processing task, in which case the processing of those partitions is interleaved on the task thread. However, the number of partitions in the input determines the job’s maximum degree of parallelism.

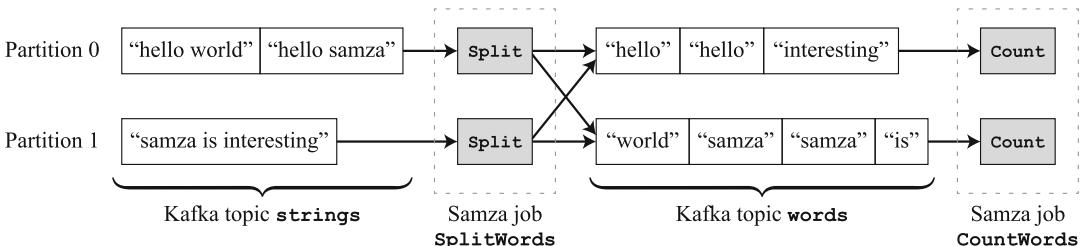
The log interface assumes that each partition of the input is a totally ordered sequence of records and that each record is associated with a monotonically increasing sequence number or identifier (known as *offset*). Since the records in each partition are read sequentially, a job can track its progress by periodically writing the offset of the last read record to durable storage. If a stream processing task is restarted, it resumes consuming the input from the last recorded offset.

Most commonly, Samza is used in conjunction with Apache Kafka (see separate article on Kafka). Kafka provides a partitioned, fault-tolerant log that allows publishers to append messages to a log partition and consumers (subscribers) to sequentially read the messages in a log partition (Wang et al. 2015; Kreps et al. 2011; Goodhope et al. 2012). Kafka also allows stream processing jobs to reprocess previously seen records by resetting the consumer offset to an earlier position, a fact that is useful during recovery from failures.

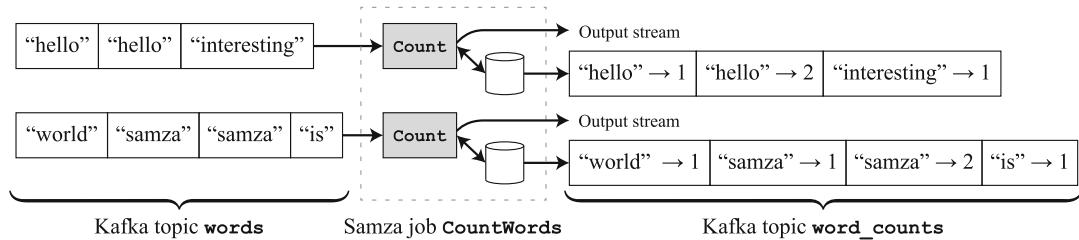
However, Samza’s stream interface is pluggable: besides Kafka, it can use any storage or messaging system as input, provided that the system can adhere to the partitioned log interface. By default, Samza can also read files from the Hadoop Distributed Filesystem (HDFS) as input, in a way that parallels MapReduce jobs, at competitive performance (Noghabi et al. 2017). At LinkedIn, Samza is commonly deployed with *Databus* inputs: Databus is a change data capture technology that records the log of writes to a database and makes this log available for applications to consume (Das et al. 2012; Qiao et al. 2013). Processing the stream of writes to a database enables jobs to maintain external indexes or materialized views onto data in a database and is especially relevant in conjunction with Samza’s support for local state (see section “Fault-Tolerant Local State”) (Fig. 3).

While every partition of an input stream is assigned to one particular task of a Samza job, the output partitions are not bound to tasks. That is, when a `StreamTask` emits output messages, it can assign them to any partition of the output stream. This fact can be used to group related data items into the same partition: for example, in the word-counting application illustrated in Fig. 2, the `SplitWords` task chooses the output partition for each word based on a hash of the word. This ensures that when different tasks encounter occurrences of the same word, they are all written to the same output partition, from where a downstream job can read and aggregate the occurrences.

When stream tasks are composed into multi-stage processing pipelines, the output of one task



Apache Samza, Fig. 2 A Samza task consumes input from one partition, but can send output to any partition (Image source: Kleppmann and Kreps 2015, © 2015 IEEE, reused with permission)



Apache Samza, Fig. 3 A task’s local state is made durable by emitting a changelog of key-value pairs to Kafka (Image source: Kleppmann and Kreps 2015, © 2015 IEEE, reused with permission)

becomes the input to another task. Unlike many other stream processing frameworks, Samza does not implement its own message transport layer to deliver messages between stream operators. Instead, Kafka is used for this purpose; since Kafka writes all messages to disk, it provides a large buffer between stages of the processing pipeline, limited only by the available disk space on the Kafka brokers.

Typically, Kafka is configured to retain several days or weeks worth of messages in each topic. Thus, if one stage of a processing pipeline fails or begins to run slow, Kafka can simply buffer the input to that stage while leaving ample time for the problem to be resolved. Unlike system designs based on backpressure, which require a producer to slow down if the consumer cannot keep up, the failure of one Samza job does not affect any upstream jobs that produce its inputs. This fact is crucial for the robust operation of large-scale systems, since it provides fault containment: as far as possible, a fault in one part of the system does not negatively impact other parts of the system.

Messages are dropped only if the failed or slow processing stage is not repaired within the retention period of the Kafka topic. In this case, dropping messages is desirable because it isolates the fault: the alternative – retaining messages indefinitely until the job is repaired – would lead to resource exhaustion (running out of memory or disk space), which would cause a cascading failure affecting unrelated parts of the system.

Thus, Samza’s design of using Kafka’s on-disk logs for message transport is a crucial factor in its scalability: in a large organization, it is often the case that an event stream produced by

one team’s job is consumed by one or more jobs that are administered by other teams. The jobs may be operating at different levels of maturity: for example, a stream produced by an important production job may be consumed by several unreliable experimental jobs. Using Kafka as a buffer between jobs ensures that adding an unreliable consumer does not negatively impact the more important jobs in the system.

Finally, an additional benefit of using Kafka for message transport is that every message stream in the system is accessible for debugging and monitoring: at any point, an additional consumer can be attached to inspect the message flow.

Fault-Tolerant Local State

Stateless stream processing, in which any message can be processed independently from any other message, is easy to implement and scale. However, many important applications require that stream processing tasks maintain state. For example:

- when performing a join between two streams, a task must maintain an index of messages seen on each input within some time window, in order to find messages matching the join condition when they arrive;
- when computing a rate (number of events per time interval) or aggregation (e.g., sum of a particular field), a task must maintain the current aggregate value and update it based on incoming events;

- when processing an event requires a database query to look up some related data (e.g., looking up a user record for the user who performed the action in the event), the database can also be regarded as stream processor state.

Many stream processing frameworks use transient state that is kept in memory in the processing task, for example, in a hash table. However, such state is lost when a task crashes or when a processing job is restarted (e.g., to deploy a new version). To make the state fault-tolerant, some frameworks such as Apache Flink periodically write checkpoints of the in-memory state to durable storage (Carbone et al. 2015); this approach is reasonable when the state is small, but it becomes expensive as the state grows (Noghabi et al. 2017).

Another approach, used, for example, by Apache Storm, is to use an external database or key-value store for any processor state that needs to be fault-tolerant. This approach carries a severe performance penalty: due to network latency, accessing a database on another node is orders of magnitude slower than accessing local in-process state (Noghabi et al. 2017). Moreover, a high-throughput stream processor can easily overwhelm the external database with queries; if the database is shared with other applications, such overload risks harming the performance of other applications to the point that they become unavailable (Kreps 2014).

In response to these problems, Samza pioneered an approach to managing state in a stream task that avoids the problems of both checkpointing and remote databases. Samza's approach to providing fault-tolerant local state has subsequently been adopted in the Kafka Streams framework (see article on Apache Kafka).

Samza allows each task to maintain state on the local disk of the processing node, with an in-memory cache for frequently accessed items. By default, Samza uses RocksDB, an embedded key-value store that is loaded into the JVM process of the stream task, but other storage engines can also be plugged in its place. In Fig. 1, the CountWords task accesses this managed state

through the `KeyValueStore` interface. For workloads with good locality, Samza's RocksDB with cache provides performance close to in-memory stores; for random-access workloads on large state, it remains significantly faster than accessing a remote database (Noghabi et al. 2017).

If a job is cleanly shut down and restarted, for example, to deploy a new version, Samza's host affinity feature tries to launch each `StreamTask` instance on the machine that has the appropriate RocksDB store on its local disk (subject to available resources). Thus, in most cases the state survives task restart without any further action. However, in some cases – for example, if a processing node suffers a full system failure – the state on the local disk may be lost or rendered inaccessible.

In order to survive the loss of local disk storage, Samza again relies on Kafka. For each store containing state of a stream task, Samza creates a Kafka topic called a *changelog* that serves as a replication log for the store. Every write to the local RocksDB store is also encoded as a message and published to this topic, as illustrated in Fig. 3. These writes can be performed asynchronously in batches, enabling much greater throughput than synchronous random-access requests to a remote data store. The write queue needs to only be flushed when the offsets of input streams are written to durable storage, as described in the last section.

When a Samza task needs to recover its state after the loss of local storage, it reads all messages in the appropriate partition of the changelog topic and applies them to a new RocksDB store. When this process completes, the result is a new copy of the store that contains the same data as the store that was lost. Since Kafka replicates all data across multiple nodes, it is suitable for fault-tolerant durable storage of this changelog.

If a stream task repeatedly writes new values for the same key in its local storage, the changelog contains many redundant messages, since only the most recent value for a given key is required in order to restore local storage. To remove this redundancy, Samza uses a Kafka feature called *log compaction*

on the changelog topic. With log compaction enabled, Kafka runs a background process that searches for messages with the same key and discards all but the most recent of those messages. Thus, whenever a key in the store is overwritten with a new value, the old value is eventually removed from the changelog. However, any key that is not overwritten is retained indefinitely by Kafka. This compaction process, which is very similar to internal processes in log-structured storage engines, ensures that the storage cost and recovery time from a changelog corresponds to the size of the state, independently of the total number of messages ever sent to the changelog (Kleppmann 2017).

Cluster-Based Task Scheduling

When a new stream processing job is started, it must be allocated computing resources: CPU cores, RAM, disk space, and network bandwidth. Those resources may need to be adjusted from time to time as load varies and reclaimed when a job is shut down.

At large organizations, hundreds or thousands of jobs need to run concurrently. At such scale, it is not practical to manually assign resources: task scheduling and resource allocation must be automated. To maximize hardware utilization, many jobs and applications are deployed to a shared pool of machines, with each multi-core machine typically running a mixture of tasks from several different jobs.

This architecture requires infrastructure for managing resources and for deploying the code of processing jobs to the machines on which it is to be run. Some frameworks, such as Storm and Flink, have built-in mechanisms for resource management and deployment. However, frameworks that perform their own task scheduling and cluster management generally require a static assignment of computing resources – potentially even dedicated machines – before any jobs can be deployed to the cluster. This static resource allocation leads to inefficiencies in machine

utilization and limits the ability to scale on demand (Kulkarni et al. 2015).

By contrast, Samza relies on existing cluster management software, which allows Samza jobs to share a pool of machines with non-Samza applications. Samza supports two modes of distributed operation:

- A job can be deployed to a cluster managed by Apache Hadoop YARN (Vavilapalli et al. 2013). YARN is a general-purpose resource scheduler and cluster manager that can run stream processors, MapReduce batch jobs, data analytics engines, and various other applications on a shared cluster. Samza jobs can be deployed to existing YARN clusters without requiring any special cluster-level configuration or resource allocation.
- Samza also supports a *stand-alone* mode in which a job’s JVM instances are deployed and executed through some external process that is not under Samza’s control. In this case, the instances use Apache ZooKeeper (Junqueira et al. 2011) to coordinate their work, such as assigning partitions of the input streams.

The stand-alone mode allows Samza to be integrated with an organization’s existing deployment and cluster management tools or with cloud computing platforms: for example, Netflix runs Samza jobs directly as EC2 instances on Amazon Web Services (AWS), relying on the existing cloud facilities for resource allocation (Paramasivam 2016). Moreover, Samza’s cluster management interface is pluggable, enabling further integrations with other technologies such as Mesos (Hindman et al. 2011).

With large deployments, an important concern is resource isolation, that is, ensuring that each process receives the resources it requested and that a misbehaving process cannot starve colocated processes of resources. When running in YARN, Samza supports the Linux cgroups feature to enforce limits on the CPU and memory use of stream processing tasks. In virtual machine environments such as EC2, resource isolation is enforced by the hypervisor.

Summary

Apache Samza is a stream processing framework that is designed to provide high throughput and operational robustness at very large scale. Efficient resource utilization requires a mixture of different jobs to share a multi-tenant computing infrastructure. In such an environment, the primary challenge in providing robust operation is fault isolation, that is, ensuring that a faulty process cannot disrupt correctly running processes and that a resource-intensive process cannot starve others.

Samza isolates stream processing jobs from each other in several different ways. By using Kafka's on-disk logs as a large buffer between producers and consumers of a stream, instead of backpressure, Samza ensures that a slow or failed consumer does not affect upstream jobs. By providing fault-tolerant local state as a common abstraction, Samza improves performance and avoids reliance on external databases that might be overloaded by high query volume. Finally, by integrating with YARN and other cluster managers, Samza builds upon existing resource scheduling and isolation technology that allows a cluster to be shared between many different applications without risking resource starvation.

Cross-References

- ▶ [Apache Flink](#)
- ▶ [Apache Kafka](#)
- ▶ [Continuous Queries](#)

References

- Calisi L (2016) How to convert legacy Hadoop Map/Reduce ETL systems to Samza streaming. <https://www.youtube.com/watch?v=KQ5OnL2hMBY>
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38. <http://sites.computer.org/debull/A15dec/p28.pdf>
- Chen S (2016) Scalable complex event processing on Samza @Uber. <https://www.slideshare.net/Shuiyichen2/scalable-complex-event-processing-on-samza-uber>
- Das S, Botev C, Surlaker K, Ghosh B, Varadarajan B, Nagaraj S, Zhang D, Gao L, Westerman J, Ganti P, Shkolnik B, Topiwala S, Pachev A, Somasundaram N, Subramaniam S (2012) All aboard the Databus! LinkedIn's scalable consistent change data capture platform. In: 3rd ACM symposium on cloud computing (SoCC). <https://doi.org/10.1145/2391229.2391247>
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: 6th USENIX symposium on operating system design and implementation (OSDI)
- Feng T (2015) Benchmarking apache Samza: 1.2 million messages per second on a single node. <http://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node>
- Goodhope K, Koshy J, Kreps J, Narkhede N, Park R, Rao J, Ye VY (2012) Building LinkedIn's real-time activity data pipeline. IEEE Data Eng Bull 35(2):33–45. <http://sites.computer.org/debull/A12june/A12JUN-CD.pdf>
- Hermann J, Balso MD (2017) Meet michelangelo: uber's machine learning platform. <https://eng.uber.com/michelangelo/>
- Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: 8th USENIX symposium on networked systems design and implementation (NSDI)
- Junqueira FP, Reed BC, Serafini M (2011) Zab: high-performance broadcast for primary-backup systems. In: 41st IEEE/IFIP international conference on dependable systems and networks (DSN), pp 245–256. <https://doi.org/10.1109/DSN.2011.5958223>
- Kleppmann M (2017) Designing data-intensive applications. O'Reilly Media. ISBN:978-1-4493-7332-0
- Kleppmann M, Kreps J (2015) Kafka, Samza and the Unix philosophy of distributed data. IEEE Data Eng Bull 38(4):4–14. <http://sites.computer.org/debull/A15dec/p4.pdf>
- Kreps J (2014) Why local state is a fundamental primitive in stream processing. <https://www.oreilly.com/ideas/why-local-state-is-a-fundamental-primitive-in-stream-processing>
- Kreps J, Narkhede N, Rao J (2011) Kafka: a distributed messaging system for log processing. In: 6th international workshop on networking meets databases (NetDB)
- Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel JM, Ramasamy K, Taneja S (2015) Twitter heron: stream processing at scale. In: ACM international conference on management of data (SIGMOD), pp 239–250. <https://doi.org/10.1145/2723372.2723374>
- Netflix Technology Blog (2016) Kafka inside Keystone pipeline. <http://techblog.netflix.com/2016/04/kafka-inside-keystone-pipeline.html>
- Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza:

- stateful scalable stream processing at LinkedIn. Proc VLDB Endow 10(12):1634–1645. <https://doi.org/10.14778/3137765.3137770>
- Paramasivam K (2016) Stream processing with Apache Samza – current and future. <https://engineering.linkedin.com/blog/2016/01/whats-new-samza>
- Pathirage M, Hyde J, Pan Y, Plale B (2016) SamzaSQL: scalable fast data management with streaming SQL. In: IEEE international workshop on high-performance big data computing (HPBDC), pp 1627–1636. <https://doi.org/10.1109/IPDPSW.2016.141>
- Qiao L, Auradar A, Beaver C, Brandt G, Gandhi M, Gopalakrishna K, Ip W, Jagadish S, Lu S, Pachev A, Ramesh A, Surlaker K, Sebastian A, Shanbhag R, Subramaniam S, Sun Y, Topiwala S, Tran C, Westerman J, Zhang D, Das S, Quiggle T, Schulman B, Ghosh B, Curtis A, Seeliger O, Zhang Z (2013) On brewing fresh Espresso: LinkedIn’s distributed data serving platform. In: ACM international conference on management of data (SIGMOD), pp 1135–1146. <https://doi.org/10.1145/2463676.2465298>
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O’Malley O, Radia S, Reed B, Baldeschwieler E (2013) Apache Hadoop YARN: yet another resource negotiator. In: 4th ACM symposium on cloud computing (SoCC). <https://doi.org/10.1145/2523616.2523633>
- Wang G, Koshy J, Subramanian S, Paramasivam K, Zadeh M, Narkhede N, Rao J, Kreps J, Stein J (2015) Building a replicated logging system with Apache Kafka. Proc VLDB Endow 8(12):1654–1655. <https://doi.org/10.14778/2824032.2824063>

Apache Spark

Alexandre da Silva Veith and Marcos Dias de Assuncao
Inria Avalon, LIP Laboratory, ENS Lyon,
University of Lyon, Lyon, France

Definitions

Apache Spark is a cluster computing solution and in-memory processing framework that extends the MapReduce model to support other types of computations such as interactive queries and stream processing (Zaharia et al. 2012). Designed to cover a variety of workloads, Spark introduces an abstraction called Resilient Distributed Datasets (RDDs) that enables running

computations in memory in a fault-tolerant manner. RDDs, which are immutable and partitioned collections of records, provide a programming interface for performing operations, such as map, filter, and join, over multiple data items. For fault-tolerance purposes, Spark records all transformations carried out to build a dataset, thus forming a *lineage graph*.

A

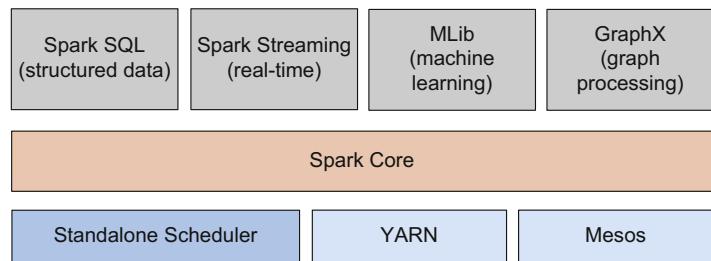
Overview

Spark (Zaharia et al. 2016) is an open-source big data framework originally developed at the University of California at Berkeley and later adopted by the Apache Foundation, which has maintained it ever since. Spark was designed to address some of the limitations of the MapReduce model, especially the need for speed processing of large datasets. By using RDDs, purposely designed to store restricted amounts of data in memory, Spark enables performing computations more efficiently than MapReduce, which runs computations on the disk.

Although the project contains multiple components, at its core (Fig. 1) Spark is a computing engine that schedules, distributes, and monitors applications comprising multiple tasks across nodes of a computing cluster (Karau et al. 2015). For cluster management, Spark supports its native Spark cluster (standalone), Apache YARN (Vavilapalli et al. 2013), or Apache Mesos (Hindman et al. 2011). At the core also lies the RDD abstraction. RDDs are sets of data items distributed across the cluster nodes and that can be manipulated in parallel. At a higher level, it provides support for multiple tightly integrated components for handling various types of workloads such as SQL, streaming, and machine learning.

Figure 2 depicts an example of a word count application using Spark’s Scala API for manipulating datasets. Spark computes RDDs in a lazy fashion, the first time they are used. Hence, the code in the example is evaluated when the counts are saved to disk, in which moment the results of the computation are required. Spark can also read data from

Apache Spark, Fig. 1
The Apache Spark stack
(Karau et al. 2015)



Apache Spark, Fig. 2
Word count example using
Spark's Scala API (Karau
et al. 2015)

```

// Create a Scala Spark configuration context
val config = new SparkConf().setAppName("WordCount")
val sc = new SparkContext(config)

// Load the input data
val input = sc.textFile(theInputFile)

// Split it into words
val words = input.flatMap(line => line.split(" "))

// Transform into pairs and count
val counts = words.map(word =>
    (word, 1)).reduceByKey{case (x, y) => x + y}

// Save the word count to a text file
counts.saveAsTextFile(theOutputFile)
  
```

various sources, such as Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift (<https://wiki.openstack.org/wiki/Swift>), and Amazon Simple Storage Service (S3) (<https://aws.amazon.com/s3/>).

Spark SQL

Spark SQL (Armbrust et al. 2015) is a module for processing structured data (<https://spark.apache.org/docs/latest/sql-programming-guide.html>). It builds on the RDD abstraction by providing Spark core engine with more information about the structure of the data and the computation being performed. In addition to enabling users to perform SQL queries, Spark SQL provides the Dataset API, which offers datasets and DataFrames. A dataset can be built using JVM objects that can then be manipulated using functional transformations. A DataFrame can be built from a large number of sources and is analogous to a table in a relational database; it is a dataset organized into named columns.

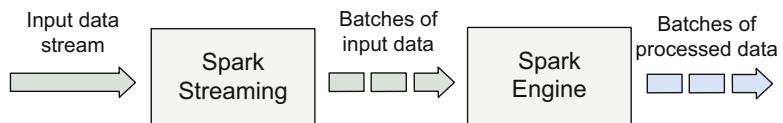
Spark Streaming

Spark Streaming provides a micro-batch-based framework for processing data streams. Data can be ingested from systems such as Apache Kafka (<https://kafka.apache.org/>), Flume, or Amazon Kinesis (<https://aws.amazon.com/kinesis/>). Under the traditional stream processing approach based on a graph of continuous operators that process tuples as they arrive (i.e., the dataflow model), it is arguably difficult to achieve fault tolerance and handle stragglers. As application state is often kept by multiple operators, fault tolerance is achieved either by replicating sections of the processing graph or via upstream backup. The former demands synchronization of operators via a protocol such as Flux (Shah et al. 2003) or other transactional protocols (Wu et al. 2015), whereas the latter, when a node fails, requires parents to replay previously sent messages to rebuild the state.

Spark Streaming uses a high-level abstraction called *discretised stream* or *DStream* (Zaharia et al. 2013). As depicted in Fig. 3, DStreams

Apache Spark, Fig. 3

High-level view of discretized streams



A

follow a micro-batch approach that organizes stream processing as batch computations carried out periodically over small time windows. During a short time interval, DStreams store the received data, which the cluster resources then use as input dataset for performing parallel computations once the interval elapses. These computations produce new datasets that represent an intermediate state or computation outputs. The intermediate state consists of RDDs that DStreams process along with the datasets stored during the next interval. In addition to providing a strong unification with batch processing, this model stores the state in memory as RDDs that DStreams can deterministically recompute. This micro-batch approach, however, sacrifices response time as the delay for processing events is dependent on the length of the micro-batches.

MLlib

Spark contains a library with common machine learning (ML) functionality such as learning algorithms for classification, regression, clustering, and collaborative filtering and featurization including feature extraction, transformation, dimensionality reduction, and selection (Meng et al. 2016). MLlib also enables the creation of ML pipelines and persistence of algorithms, models, and pipelines. These features are designed to scale out across large computing clusters using Spark’s core engine.

GraphX

GraphX (Gonzalez et al. 2014) extends the RDD API by enabling the creation of a multigraph (i.e., the property graph) with arbitrary properties attached to vertices and edges. The library is designed for manipulating graphs, exposing a set of operators (e.g., subgraph, joinVertices), and for carrying out parallel computations. It contains a library of common graph algorithms, such as PageRank and triangle counting.

Examples of Applications

Spark has been used for several data processing and data science tasks, but the range of applications that it enables is endless. Freeman et al. (2014), for instance, designed a library called Thunder on top of Spark for large-scale analysis of neural data. Many machine learning and statistical algorithms have been implemented for MLlib, which simplifies the construction of machine learning pipelines. The source code of Spark has also grown substantially since it became an Apache project. Numerous third-party libraries and packages have been included for performing tasks in certain domains or for simplifying the use of existing APIs.

Spark provides the functionalities that data scientists need to perform data transformation, processing, and analysis. Data scientists often need to perform ad hoc exploration during which they have to test new algorithms or verify the results in the least amount of time. Spark provides APIs, libraries, and shells that allow scientists to perform such tasks while enabling them to test their algorithms on large problem sizes. Once the exploration phase is performed, the solution is productized by engineers who integrate the data analysis tasks into an often more complex business application.

Examples of applications built using Apache Spark include analysis of data from mobile devices (Alsheikh et al. 2016) and Internet of Things (IoT), web-scale graph analytics, anomaly detection of user behavior and network traffic for information security (Ryza et al. 2017), real-time machine learning, data stream processing pipelines, engineering workloads, and geospatial data processing, to cite just a few. New application scenarios are presented each year during Spark’s Summit (<https://spark-summit.org/>), an event that has become a showcase of next-generation big data applications.

Future Directions of Research

Spark APIs shine both during exploratory work and when engineering a solution deployed in production. Over the years, much effort has been paid toward making APIs easier to use and to optimize, for instance, the introduction of the Dataset API to avoid certain performance degradations that could occur if a user did not properly design the chain of operations executed by the Spark engine. As mentioned earlier, considerable work has also focused on creating new libraries and packages, including for processing live streams. The discretized model employed by Spark's stream processing API, however, introduces some delays depending on the time length of micro-batches. More recent and emerging application scenarios, such as analysis of vehicular traffic and networks, monitoring of operational infrastructure, wearable assistance (Ha et al. 2014), and 5G services, require data processing and service response under very short delays. Spark can be used as part of a larger service workflow, but alone it does not provide means to address some of the challenging scenarios that require very short response times. This requires the use of other frameworks such as Apache Storm.

To reduce the latency of applications delivered to users, many service components are also increasingly being deployed at the edges of the Internet (Hu et al. 2015) under a model commonly called edge computing. Some frameworks are available for processing streams of data using resources at the edge (e.g., Apache Edgent (<https://edgent.apache.org/>)), whereas others are emerging. There are also frameworks that aim to provide high-level programming abstractions for creating dataflows (e.g., Apache Beam (<https://beam.apache.org/>)) with underlying execution engines for several processing solutions. There is, however, a lack of unifying solutions on programming models and abstractions for exploring resources from both cloud and edge computing deployments, as well as scheduling and resource management tools for deciding what processing tasks need to be offloaded to the edge.

Cross-References

- ▶ Cloud Computing for Big Data Analysis
- ▶ Definition of Data Streams
- ▶ Graph Processing Frameworks
- ▶ Hadoop
- ▶ Large-Scale Graph Processing System
- ▶ Programmable Memory/Processing in Memory (PIM)
- ▶ Streaming Microservices

References

- Alsheikh MA, Niyato D, Lin S, Tan H-P, Han Z (2016) Mobile big data analytics using deep learning and Apache Spark. *IEEE Netw* 30(3):22–29
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in Spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data (SIGMOD'15). ACM, New York, pp 1383–1394
- Freeman J, Vladimirov N, Kawashima T, Mu Y, Sofroniew NJ, Bennett DV, Rosen J, Yang C-T, Looger LL, Ahrens MB (2014) Mapping brain activity at scale with cluster computing. *Nat Methods* 11(9):941–950
- Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. In: OSDI, vol 14, pp 599–613
- Hu K, Chen Z, Hu W, Richter W, Pillai P, Satyanarayanan M (2014) Towards wearable cognitive assistance. In: 12th annual international conference on mobile systems, applications, and services, MobiSys'14. ACM, New York, pp 68–81. <https://doi.org/10.1145/2594368.2594383>
- Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI, vol 11, pp 22–22
- Hu YC, Patel M, Sabella D, Sprecher N, Young V (2015) Mobile edge computing – a key technology towards 5G. ETSI White Paper 11(11):1–16
- Karau H, Konwinski A, Wendell P, Zaharia M (2015) Learning Spark: lightning-fast big data analysis. O'Reilly Media, Inc., Beijing
- Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (2016) Mllib: machine learning in Apache Spark. *J Mach Learn Res* 17(1):1235–1241
- Ryza S, Laserson U, Owen S, Wills J (2017) Advanced analytics with Spark: patterns for learning from data at scale. O'Reilly Media, Inc., Sebastopol
- Shah MA, Hellerstein JM, Chandrasekaran S, Franklin MJ (2003) Flux: an adaptive partitioning operator

- for continuous query systems. In: 19th international conference on data engineering (ICDE 2003). IEEE Computer Society, pp 25–36
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O’Malley O, Radia S, Reed B, Baldeschwieler E (2013) Apache hadoop YARN: yet another resource negotiator. In: 4th annual symposium on cloud computing (SOCC’13). ACM, New York, pp 5:1–5:16. <https://doi.org/10.1145/2523616.2523633>
- Wu Y, Tan KL (2015) ChronoStream: elastic stateful stream computation in the cloud. In: 2015 IEEE 31st international conference on data engineering, pp 723–734
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: 9th USENIX conference on networked systems design and implementation (NSDI’12). USENIX Association, Berkeley, pp 2–2
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: 24th ACM symposium on operating systems principles (SOSP’13). ACM, New York, pp 423–438
- Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I (2016) Apache Spark: a unified engine for big data processing. Commun ACM 59(11):56–65

Apache Spark Benchmarking

► [SparkBench](#)

Apache SystemML

Declarative Large-Scale Machine Learning

Matthias Boehm
IBM Research – Almaden, San Jose, CA, USA

Definitions

Apache SystemML (Ghoting et al. 2011; Boehm et al. 2016) is a system for declarative, large-scale machine learning (ML) that aims to increase the

productivity of data scientists. ML algorithms are expressed in a high-level language with R- or Python-like syntax, and the system automatically generates efficient, hybrid execution plans of single-node CPU or GPU operations, as well as distributed operations using data-parallel frameworks such as MapReduce (Dean and Ghemawat 2004) or Spark (Zaharia et al. 2012). SystemML’s high-level abstraction provides the necessary flexibility to specify custom ML algorithms while ensuring physical data independence, independence of the underlying runtime operations and technology stack, and scalability for large data. Separating the concerns of algorithm semantics and execution plan generation is essential for the automatic optimization of execution plans regarding different data and cluster characteristics, without the need for algorithm modifications in different deployments.

A

Overview

In SystemML (Ghoting et al. 2011; Boehm et al. 2016), data scientists specify their ML algorithms using a language with R- or Python-like syntax. This language supports abstract data types for scalars, matrices and frames, and operations such as linear algebra, element-wise operations, aggregations, indexing, and statistical operations but also control structures such as loops, branches, and functions. These scripts are parsed into a hierarchy of statement blocks and statements, where control flow delineates the individual blocks. For each block of statements, the system then compiles DAGs (directed acyclic graphs) of high-level operators (HOPs), which is the core internal representation of SystemML’s compiler. Size information such as matrix dimensions and sparsity are propagated via intra- and inter-procedural analysis from the inputs through the entire program. This size information is then used to compute memory estimates per operator and accordingly select physical operators resulting in a DAG of low-level operators (LOPs). These LOP DAGs are finally compiled into executable instructions.

Example: As an example, consider the following script for linear regression via a closed-

form method that computes and solves the normal equations:

```

1: X = read($X);
2: y = read($Y);
3: lambda = 0.001;
4: if( $icpt == 1 )
5:     X = cbind(X, matrix(1, nrow(X), 1));
6: I = matrix(1, ncol(X), 1);
7: A = t(X) %*% X + diag(I) * lambda;
8: b = t(X) %*% y;
9: beta = solve(A, b);
10: write(beta, $B);

```

This script reads the feature matrix \mathbf{X} and labels \mathbf{y} , optionally appends a column of 1s to \mathbf{X} for computing the intercept, computes the normal equations, and finally solves the resulting linear system of equations. SystemML then compiles, for example, for lines 6–10, a HOP DAG that contains logical operators such as matrix multiplications for $\mathbf{X}^\top \mathbf{X}$ and $\mathbf{X}^\top \mathbf{y}$. Given input metadata (e.g., let \mathbf{X} be a dense $10^7 \times 10^3$ matrix), the compiler also computes memory estimates for each operation (e.g., 80.08 GB for $\mathbf{X}^\top \mathbf{y}$). If the memory estimate of an operation exceeds the driver memory budget, this operation is scheduled for distributed execution, and appropriate physical operators are selected (e.g., mapmm as a broadcast-based operator for $\mathbf{X}^\top \mathbf{y}$).

Static and Dynamic Rewrites: SystemML’s optimizer applies a broad range of optimizations throughout its compilation chain. An important class of optimizations with high-performance impact are rewrites. SystemML applies static and dynamic rewrites (Boehm et al. 2014a). Static rewrites are size-independent and include traditional programming language techniques – such as common subexpression elimination, constant folding, and branch removal – algebraic simplifications for linear algebra, as well as backend-specific transformations such as caching and partitioning directives for Spark. For instance, in the above example, after constant propagation, constant folding,

and branch removal, the block of lines 4–5 is removed if $\$icpt==0$, which further allows unconditional size propagation and the merge of the entire program into a single HOP DAG. Furthermore, the expression ‘ $\text{diag}(\text{matrix}(1, \text{ncol}(\mathbf{X}), 1)) \odot \lambda$ ’ is simplified to ‘ $\text{diag}(\text{matrix}(\lambda, \text{ncol}(\mathbf{X}), 1))$ ’, which avoids unnecessary operations and intermediates. SystemML’s rewrite system contains hundreds of such rewrites, some of which even change the asymptotic behavior (e.g., $\text{trace}(\mathbf{XY}) \rightarrow \sum(\mathbf{X} \odot \mathbf{Y}^\top)$). Additional dynamic rewrites are size-dependent because they require sizes for cost estimation or validity constraints. Examples are matrix multiplication chain optimization as well as dynamic simplification rewrites such as $\sum(\mathbf{X}^2) \rightarrow \mathbf{X}^\top \mathbf{X} \mid \text{ncol}(\mathbf{X}) = 1$. The former exploits the associativity of matrix multiplications and aims to find an optimal parenthesization for which SystemML applies a textbook dynamic programming algorithm (Boehm et al. 2014a).

Operator Selection and Fused Operators: Another important class of optimizations is the selection of execution types and physical operators (Boehm et al. 2016). SystemML’s optimizer analyzes the memory budgets of the driver and executors and selects – based on worst-case memory estimates (Boehm et al. 2014b) – local or distributed execution types. Besides data and cluster characteristics (e.g., data size/shape, memory,

and parallelism), the compiler also considers matrix and operation properties (e.g., diagonal/symmetric matrices, sparse-safe operations) as well as data flow properties (e.g., co-partitioning and data locality). Depending on the chosen execution types, different physical operators are considered with a selection preference from local or special-purpose to shuffle-based operators. For example, the multiplication $\mathbf{X}^\top \mathbf{X}$ from line 7 allows for a special transpose-self operator (`t smm`), which is an easily parallelizable unary operator that exploits the output symmetry for less computation. For distributed operations, this operator has a block size constraint because it requires access to entire rows. If special-purpose or broadcast-based operators do not apply, the compiler falls back to the shuffle-based `cpmm` and `rmm` operators (Ghoting et al. 2011). Additionally, SystemML replaces special patterns with hand-coded fused operators to avoid unnecessary intermediates (Huang et al. 2015; Elgamal et al. 2017) and unnecessary scans (Boehm et al. 2014a; Ashari et al. 2015; Elgamal et al. 2017), as well as to exploit sparsity across chains of operations (Boehm et al. 2016; Elgamal et al. 2017). For example, computing the weighted squared loss via $\text{sum}(\mathbf{W} \odot (\mathbf{X} - \mathbf{U}\mathbf{V}^\top)^2)$ would create huge dense intermediates. In contrast, sparsity-exploiting operators leverage the sparse driver (i.e., the sparse matrix \mathbf{W} and the sparse-safe multiply \odot) for selective computation that only considers nonzeros in \mathbf{W} .

Dynamic Recompilation: Dynamic rewrites and operator selection rely on size information for memory estimates, cost estimation, and validity constraints. Hence, unknown dimensions or sparsity lead to conservative fallback plans. Example scenarios are complex conditional control flow or function call graphs, user-defined functions, data-dependent operations, computed size expressions, and changing sizes or sparsity as shown in the following example:

```

1: while( continue ) {
2:     parfor( i in 1:n ) {
3:         if( fixed[1,i] == 0 ) {
4:             X = cbind(Xg, Xorig[,i]);
5:             AIC[1,i] = linregDS(X,y);
6:         }
7:         #select & append best to Xg
8:     }
9: }
```

This example originates from a stepwise linear regression algorithm for feature selection that iteratively selects additional features and calls the previously introduced regression algorithm. SystemML addresses this challenge of unknown sizes via dynamic recompilation (Boehm et al. 2014a) that recompiles subplans – at the granularity of HOP DAGs – with exact size information of intermediates during runtime. During initial compilation, operations and DAGs with unknowns are marked for dynamic recompilation, which also includes splitting DAGs after data-dependent operators. During runtime, the recompiler then deep copies the HOP DAG, updates sizes, applies dynamic rewrites, recomputes memory estimates, generates new runtime instructions, and resumes execution with the intermediate results computed so far. This approach yields good plans and performance even in the presence of initial unknowns.

Runtime Integration: At runtime level, SystemML interprets the generated instructions. Single-node and Spark operations directly map to instructions, whereas for the MapReduce backend, instructions are packed into a minimal number of MR jobs. For distributed operations, matrices (and similarly frames) are stored in a blocked representation of pairs of block indexes and blocks with fixed block size, where individual blocks can be dense, sparse, or ultra-sparse. In contrast, for single-node operations, the entire matrix is represented as a single block, which allows reusing the block runtime across backends. Data transfers between the local and distributed backends and driver memory management are handled by a multilevel buffer pool (Boehm et al. 2016) that controls local evictions, parallelizes and collects RDDs, creates broadcasts, and reads/writes data from/to the distributed file system. For example, a single-node instruction first pins its inputs into memory – which triggers reads from HDFS or RDDs if necessary – performs the block operation, registers the output in the buffer pool, and finally unpins its inputs. Many block operations and the I/O system for different formats are multi-threaded to exploit parallelism in scale-up environments. For compute-intensive deep

learning workloads, SystemML further calls native CPU and GPU libraries for BLAS and DNN operations. In contrast to other deep learning frameworks, SystemML also supports sparse neural network operations. Memory management for the GPU device is integrated with the buffer pool allowing for lazy data transfer on demand. Finally, SystemML uses numerically stable operations based on Kahan addition for descriptive statistics and certain aggregation functions (Tian et al. 2012).

Key Research Findings

In addition to the compiler and runtime techniques described so far, there are several advanced techniques with high-performance impact.

Task-Parallel Parfor Loops: SystemML’s primary focus is data parallelism. However, there are many use cases such as ensemble learning, cross validation, hyper-parameter tuning, and complex models with disjoint or overlapping data that are naturally expressed in a task-parallel manner. These scenarios are addressed by SystemML’s `parfor` construct for parallel for loops (Boehm et al. 2014b). In contrast to similar constructs in high-performance computing (HPC), `parfor` only asserts the independence of iterations, and a dedicated `parfor` optimizer reasons about hybrid parallelization strategies that combine data and task parallelism. Reconsider the stepwise linear regression example. Alternative plan choices include (1) a local, i.e., multi-threaded, `parfor` with local operations, (2) a remote `parfor` that runs the entire loop as a distributed Spark job, or (3) a local `parfor` with concurrent data-parallel Spark operations if the data does not fit into the driver.

Resource Optimization: The selection of execution types and operators is strongly influenced by memory budgets of the driver and executor processes. Finding a good static cluster configuration that works well for a broad range of ML algorithms and data sizes is a hard problem.

SystemML addresses this challenge with a dedicated resource optimizer for automatic resource provisioning (Huang et al. 2015) on resource negotiation frameworks such as YARN or Mesos. The key idea is to optimize resource configurations via an online what-if analysis with regard to the given ML program as well as data and cluster characteristics. This framework optimizes performance without unnecessary over-provisioning, which can increase throughout in shared on-premise clusters and save money in cloud environments.

Compressed Linear Algebra: Furthermore, there is a broad class of iterative ML algorithms that use repeated read-only data access and I/O-bound matrix-vector multiplications to converge to an optimal model. For these algorithms, it is crucial for performance to fit the data into available single-node or distributed memory. However, general-purpose, lightweight, and heavyweight compression techniques struggle to achieve both good compression ratios and fast decompression to enable block-wise uncompressed operations. Compressed linear algebra (CLA) (Elgohary et al. 2016) tackles this challenge by applying lightweight database compression techniques – for column-wise compression with heterogeneous encoding formats and co-coding – to matrices and executing linear algebra operations such as matrix-vector multiplications directly on the compressed representation. CLA yields compression ratios similar to heavyweight compression and thus allows fitting large datasets into memory while achieving operation performance close to the uncompressed case.

Automatic Operator Fusion: Similar to query compilation and loop fusion in databases and HPC, the opportunities for fused operators – in terms of fused chains of operations – are ubiquitous. Example benefits are a reduced number of intermediates, reduced number of scans, and sparsity exploitation across operations. Despite their high-performance impact, hand-coded fused operators are usually limited to few operators and incur a large development effort. Automatic operator fusion via code generation (Elgamal et al. 2017) overcomes this challenge

by automatically determining valid fusion plans and generating access-pattern-aware operators in the form of hand-coded skeletons with custom body code. In contrast to existing work on operator fusion, SystemML introduced a cost-based optimizer framework to find optimal fusion plans in DAGs of linear algebra programs for dense, sparse, and compressed data as well as local and distributed operations.

Examples of Application

SystemML has been applied in a variety of ML applications including statistics, classification, regression, clustering, matrix factorization, survival analysis, and deep learning. In contrast to specialized systems for graphs like GraphLab (Low et al. 2012) or deep learning like TensorFlow (Abadi et al. 2016), SystemML provides a unified system for small- to large-scale problems with support for dense, sparse, and ultra-sparse data, as well as local and distributed operations. Accordingly, SystemML’s primary application area is an environment with diverse algorithms, varying data characteristics, or different deployments.

Example deployments include large-scale computation on top of MapReduce or Spark and programmatic APIs for notebook environments or embedded scoring. Thanks to deployment-specific compilation, ML algorithms can be reused without script changes. This flexibility enabled the integration of SystemML into systems with different architectures. For example, SystemML has been shipped as part of the open-source project R4ML and the IBM products BigInsights, Data Science Experience (DSX), and multiple Watson services.

Future Directions for Research

Given the goal of a unified system for ML applications and recent algorithm and hardware trends, there are many directions for future research

throughout the stack of SystemML and similar systems (Kumar et al. 2017):

Specification Languages: SystemML focuses on optimizing fixed algorithm specifications. However, end-to-end applications would further benefit from even higher levels of abstractions for feature engineering, model selection, and life cycle management in general. A promising direction is a stack of declarative languages that allows for reuse and cross-level optimization.

Optimization Techniques: Regarding the automatic optimization of ML programs, further work is required regarding size and sparsity estimates, adaptive query processing and storage (as an extension of dynamic recompilation), and principled approaches to automatic rewrites and automatic operator fusion.

Runtime Techniques: A better support for deep learning and scientific applications requires the extension from matrices to dense/sparse tensors of different data types and their operations. Additionally, further research is required regarding the automatic exploitation of accelerators and heterogenous hardware.

Benchmarks: Finally, SystemML – but also the community at large – would benefit from dedicated benchmarks for the different classes of ML workloads and different levels of specification languages.

Cross-References

- ▶ [Apache Mahout](#)
- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Hadoop](#)
- ▶ [Python](#)
- ▶ [Scalable Architectures for Big Data Analysis](#)
- ▶ [Tools and Libraries for Big Data Analysis](#)

References

- Abadi M et al (2016) TensorFlow: a system for large-scale machine learning. In: OSDI

- Ashari A, Tatikonda S, Boehm M, Reinwald B, Campbell K, Keenleyside J, Sadayappan P (2015) On optimizing machine learning workloads via Kernel fusion. In: PPoPP
- Boehm M, Burdick DR, Evfimievski AV, Reinwald B, Reiss FR, Sen P, Tatikonda S, Tian Y (2014a) SystemML's optimizer: plan generation for large-scale machine learning programs. *IEEE Data Eng Bull* 37(3):52–62
- Boehm M, Tatikonda S, Reinwald B, Sen P, Tian Y, Burdick D, Vaithyanathan S (2014b) Hybrid parallelization strategies for large-scale machine learning in SystemML. *PVLDB* 7(7):553–564
- Boehm M, Dusenberry M, Eriksson D, Evfimievski AV, Manshadji FM, Pansare N, Reinwald B, Reiss F, Sen P, Surve A, Tatikonda S (2016) SystemML: declarative machine learning on spark. *PVLDB* 9(13): 1425–1436
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: OSDI
- Elgamal T, Luo S, Boehm M, Evfimievski AV, Tatikonda S, Reinwald B, Sen P (2017) SPOOF: sum-product optimization and operator fusion for large-scale machine learning. In: CIDR
- Elgohary A, Boehm M, Haas PJ, Reiss FR, Reinwald B (2016) Compressed linear algebra for large-scale machine learning. *PVLDB* 9(12): 960–971
- Ghoting A, Krishnamurthy R, Pednault EPD, Reinwald B, Sindhwani V, Tatikonda S, Tian Y, Vaithyanathan S (2011) SystemML: declarative machine learning on MapReduce. In: ICDE
- Huang B, Boehm M, Tian Y, Reinwald B, Tatikonda S, Reiss FR (2015) Resource elasticity for large-scale machine learning. In: SIGMOD
- Kumar A, Boehm M, Yang J (2017) Data management in machine learning: challenges, techniques, and systems. In: SIGMOD
- Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM (2012) Distributed GraphLab: a framework for machine learning in the cloud. *PVLDB* 5(8)
- Tian Y, Tatikonda S, Reinwald B (2012) Scalable and numerically stable descriptive statistics in SystemML. In: ICDE
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI

Applications

► Big Data in Computer Network Monitoring

Applications of Big Spatial Data: Health

Lance A. Waller

Department of Biostatistics and Bioinformatics,
Rollins School of Public Health, Emory
University, Atlanta, GA, USA

Definitions

The term “big spatial data” encompasses all types of big data with the addition of geographic reference information, typically a location associated with a point in space (e.g., latitude, longitude, and altitude coordinates), an area (e.g., a country, a district, or a census enumeration zone), a line or curve (e.g., a river or a road), or a pixel (e.g., high-resolution satellite images or a biomedical imaging scan). When applied to questions of health, big spatial data can aid in attempts to understand geographic variations in the risks and rates of disease (e.g., is risk here greater than risk there?), to identify local factors driving geographic variations in risks and rates (e.g., does local nutritional status impact local childhood mortality?), and to evaluate the impact of local health policies (e.g., district-specific adjustments to insurance reimbursements).

In addition to defining big spatial data, it is also important to define what is meant by “health.” The World Health Organization defines *health* as “a state of complete physical, mental and social well-being and not merely the absence of disease or infirmity” (Preamble to the WHO Constitution, <http://www.who.int/suggestions/faq/en/>). This can be interpreted at the individual level and at the population level. At the population level, the US Centers for Disease Control and Prevention define *public health* as “the science of protecting and improving the health of people and their communities. This work is achieved by promoting healthy lifestyles, researching disease and injury prevention, and detecting, preventing and responding to

infectious diseases” (<https://www.cdcfoundation.org/what-public-health>). More recently, there has been increasing interest in the concept of *population health*, i.e., “the health outcomes of a group of individuals, including the distribution of such outcomes within the group” (Kindig and Stoddard 2003). Both public and population health focus on measures of health in an overall community with public health typically focusing on health promotion and disease prevention and population health as defining current and target distributions of health outcomes across a population, typically, to evaluate the impact of policy (e.g., government funding or insurance coverage).

Overview

The rapid expansion of measurement technology, data storage, informatics, and analytics feeds rapid growth in applications of big data within the fields of individual, population, and public health. The sequencing of the human genome in 2001 was rapidly followed by the ability to measure multiple aspects of the human metabolome, transcriptome, microbiome, and overall exposure to the environment (the exposome) (Miller and Jones 2014; Wild 2005). These detailed views of the complex world of each individual’s genetic, environmental, and social makeup provide quantitative insight into a more nuanced view of “health” than simply the absence of disease; rather, the measures begin to provide a view into *how* health works at individual, local, societal, and global levels. The measures move from tracking individual biomarkers to panels of biomarkers and toward integration of heterogeneous streaming of information from wearable sensors, local exposure monitors, and georeferenced tracking devices. These heterogeneous data sources enable increasingly detailed exploration of associations between an individual’s location and their phenotype, genotype, and multiple influencing factors relating to diet, activity level, environment, and prevention activities, some of which vary locally themselves. More specifically,

the multiple measures provide insight into aspects of and interactions between the elements of individual, public, and population health, defined above.

Geography provides a key context for all three types of health; specifically, the locations where one lives and works contribute greatly to one’s individual exposome and to the range of environmental and social determinants of disease (and health) associated with the health of the local population. As a result, big spatial data play a key (and expanding) role in the definition, description, and understanding of health.

Key Research Findings

The rapid increase of health-related data and the continued growth of geographic information systems and science (Goodchild 1992) impact both health care and health research. In addition, the ecosocial theory of social determinants of health (Krieger 2001) motivated many health leaders (including the Director of the Robert Wood Johnson Foundation and the Director of the US National Institutes of Health) to comment that an individual’s ZIP code (US postal code) is a more powerful predictor of health than that individual’s genetic code.

Spatial analysis of health traditionally includes geographically referenced, local information on health outcomes, demographics, economics, policies, and potential exposures. Analyses often focus on predicting exposures from a fixed set of observations, assessing of whether point locations of disease occur in geographically concentrated “clusters” or quantifying associations between health outcomes and local measures of potential risk factors (Waller and Gotway 2004). A variety of biostatistical and epidemiologic methods have been extended to allow for potential spatial and spatiotemporal correlations.

The incorporation of big data into a spatial setting provides intriguing opportunities to refine both concepts of health and concepts of location as illustrated below.

Examples of Application

Beginning at the individual patient level, electronic health records (EHRs) contain a potential trove of medical information regarding individual's interaction with the healthcare system, consolidating information invaluable for health care (Murdoch and Detsky 2013). This information typically is stored in multiple, heterogeneous formats designed to allow access by healthcare professionals for determining patient care. Within a given health system, these records are often consolidated into secure data warehouses to allow system-wide assessment of quality of care, monitoring of costs, and reimbursement by health insurance systems. Moving from health care of an individual to research often requires assessing information from multiple systems in different formats. Recent years have seen an increase in explorations of the research potential of EHRs, including (but not limited to) applications of natural language processing of text fields filled by healthcare professionals, development of searchable database of biomedical images, and system-wide assessments of healthcare performance.

In addition to the development of informatics and analytics for EHRs, there is also a rapid increase in applications of informatics and distributed computing services to manage, merge, search, and summarize streaming data from multiple devices within a healthcare setting such as an intensive care unit, emergency department, or surgery facility. Some measures are monitored almost continuously (e.g., blood pressure, blood oxygen levels), while others occur on less frequent timescales (e.g., baseline levels, treatment protocols, caregivers on call). Similar issues arise outside of the healthcare setting in the increasing popularity of wearable monitoring technology providing feedback on activity levels, heart rate, and, in some cases, measures of one's ambient environment. In many cases, such streaming technologies are linked to mobile devices that are also recording location information leading to the development of georeferenced mobile health (mHealth) platforms and analytics (Estrin and Sim 2010; Nilsen et al. 2012).

Recent years have also seen a rise in the amount of, connectivity between, and use of

heterogeneous data from disparate sources to address health questions. Examples include (but again are not limited to) the use of remote sensing data to assess exposure potential for airborne pollutants on local (Liu et al. 2005) and global scales (Shaddick et al. 2017), the use of Internet information in public health surveillance (Brownstein et al. 2009), and the expanded use of administrative data across multiple countries coupled with reported epidemiologic associations to provide assessments of the global burden of disease (Murray and Lopez 1997). The potential is remarkable, but desire for accurate health information can also lead to the temptation to ignore past epidemiologic findings in light of the promise of predictive analytics, a temptation that can lead to gross inaccuracies and potential false alarms (Lazar et al. 2014).

In addition to expanded possibilities of referencing individual-level attributions, volunteered geographic information via location-based apps and related services provide near real-time access to individual-level movement, providing novel opportunities for assessing exposures (Sui et al. 2013). Such information on location and movement provide additional context for moving from maps of local risk (Kitron 1998) to maps of an individual's cumulative experienced risk in moving through such an exposure space. Vazquez-Prokopec et al. (2009) provide a creative example exploring space-time exposures of taxi drivers to mosquitos carrying dengue virus in Iquitos, Peru.

The expanding availabilities of individual-level health and location data raise ongoing discussions of data privacy (determined by an individual's choices of which of their individual data to reveal), data confidentiality (the ability of the data holders to maintain each individual's privacy), and data access (determining which agencies, healthcare systems, physicians can access an individual's personal information). Concepts and comfort levels relating to data privacy vary culturally and between countries leading to differences in regulatory processes to evaluate whether data holders maintain confidentiality. For example, in the United States, the Health Insurance Portability and Accountability Act (HIPAA) of 1996 defines data protection standards for "protected health

information” (PHI) and its use with the healthcare system. HIPAA lists individual location data at a finer resolution than broadscale (3-digit) US ZIP codes among its list of PHI data elements, requiring care and documentation in the collection, storage, visualization, and reporting of geographic information at this and finer resolutions.

The competitive environment of US healthcare systems and the proprietary environment of providers of electronic health record (EHR) systems result in considerable informatics challenges when individual patients interact with multiple providers across different units. Some interesting progress has been made toward informatics solutions (e.g., the Fast Healthcare Interoperability Resources (FHIR)) to address these administrative barriers to data sharing and allow data sharing through app-based platforms (Mandel et al. 2016), but this remains an active area of research in the fields of medical informatics, particularly for FHIR extensions involving geolocation data extensions.

Future Directions for Research

The examples above illustrate that health applications of big spatial data technologies draw from traditional spatial analyses within individual, population, and public health but also illustrate that, to date, such work typically occurs in application-specific settings. The development of cohesive concepts and associated accurate and reliable toolboxes for spatial analytics and their health applications remains an area of active research. These developments include many challenging issues relating to accurate healthcare monitoring, assessment, and provision for each individual patient. Health applications extend well beyond that of individual health, and provide detailed insight into population and public health as well. There is great need for creativity in informatics to allow connection between, linkage across, and searches of elements of spatially referenced health and health-related measurements. As noted above, “health-related” can range from within-individual measures of immune response to satellite-based assessments

of local environmental conditions or global models of climate impact on pollution levels or food sources. Robust and reproducible spatial health analysis requires coordination of expert knowledge across multiple disciplines in order to fully reach its vast potential.

Cross-References

- ▶ [Big Data for Health](#)
- ▶ [Privacy-Preserving Record Linkage](#)
- ▶ [Spatio-Social Data](#)
- ▶ [Spatiotemporal Data: Trajectories](#)
- ▶ [Using Big Spatial Data for Planning User Mobility](#)

References

- Brownstein JS, Freifeld CC, Madoff LC (2009) Digital disease detection: harnessing the web for public health surveillance. *N Engl J Med* 360:2153–2157
- Estrin D, Sim I (2010) Open mHealth architecture: an engine for health care innovation. *Science* 330: 759–760
- Goodchild MF (1992) Geographic information science. *Int J Geogr Inf Syst* 6:31–45
- Kindig D, Stoddart G (2003) What is population health? *Am J Public Health* 93:380–383
- Kitron U (1998) Landscape ecology and epidemiology of vector-borne diseases: tools for spatial analysis. *J Med Entomol* 35:435–445
- Krieger N (2001) Theories for social epidemiology in the 21st century: an ecosocial perspective. *Int J Epidemiol* 30:668–677
- Lazar D, Kennedy R, King G, Vespiagnani A (2014) The parable of Google Flu: traps in big data analysis. *Science* 343:1203–1205
- Liu Y, Sarnat JA, Kilaru V, Jacob DJ, Koutrakis P (2005) Estimating ground-level PM_{2.5} in the Eastern United States using satellite remote sensing. *Environ Sci Technol* 39:3269–3278
- Mandel JC, Kreda DA, Mandl KD, Kohane IS, Romoni RB (2016) SMART on FHIR: a standards-based, interoperable apps platform for electronic health records. *J Am Med Inform Assoc* 23:899–908
- Miller GM, Jones DP (2014) The nature of nurture: refining the definition of the exposome. *Toxicol Sci* 137:1–2
- Murdoch TB, Detsky AS (2013) The inevitable application of big data to health care. *J Am Med Assoc* 309:1351–1352
- Murray CJL, Lopez AD (1997) Alternative projections of mortality and disability by cause 1990–2020: Global Burden of Disease Study. *Lancet* 349:1498–1504

- Nilsen W, Kumar S, Shar A, Varoquiers C, Wiley T, Riley WT, Pavel M, Atienza AA (2012) Advancing the science of mHealth. *J Health Commun* 17(supplement 1):5–10
- Shaddick G, Thomas ML, Green A, Brauer M, van Donkelaar A, Burnett R, Chang HH, Cohen A, van Dingenen R, Dora C, Gumy S, Liu Y, Martin R, Waller LA, West J, Zidek JV, Pruss-Ustun A (2017) Data integration model for air quality: a hierarchical approach to the global estimation of exposures to air pollution. *J R Stat Soc Ser C* 67:231–253
- Sui D, Elwood S, Goodchild M (eds) (2013) Crowdsourcing geographic knowledge: volunteered geographic information in theory and practice. Springer, Dordrecht
- Vazquez-Prokopec GM, Stoddard ST, Paz-Soldan V, Morrison AC, Elder JP, Kochel TJ, Scott TW, Kitron U (2009) Usefulness of commercially available GPS data-loggers for tracking human movement and exposure to dengue virus. *Int J Health Geogr* 8:68. <https://doi.org/10.1186/1476-072X-8-68>
- Waller LA, Gotway CA (2004) Applied spatial statistics for public health data. Wiley, Hoboken
- Wild CP (2005) Complementing the genome with the “exposome”: the outstanding challenge of environmental exposure measurement in molecular epidemiology. *Cancer Epidemiol Biomark Prev* 14:1847–1850

Approximate Computation

► Tabular Computation

Approximate Computing for Stream Analytics

Do Le Quoc¹, Ruichuan Chen^{2,4}, Pramod Bhatotia³, Christof Fetzer¹, Volker Hilt², and Thorsten Strufe¹

¹TU Dresden, Dresden, Germany

²Nokia Bell Labs, Stuttgart, Germany

³The University of Edinburgh and Alan Turing Institute, Edinburgh, UK

⁴Nokia Bell Labs, NJ, USA

Introduction

Stream analytics systems are extensively used in the context of modern online services to transform continuously arriving raw data streams into useful insights (Foundation 2017a; Murray et al.

2013; Zaharia et al. 2013). These systems target low-latency execution environments with strict service-level agreements (SLAs) for processing the input data streams.

In the current deployments, the low-latency requirement is usually achieved by employing more computing resources. Since most stream processing systems adopt a data-parallel programming model (Dean and Ghemawat 2004), almost linear scalability can be achieved with increased computing resources (Quoc et al. 2013, 2014, 2015a,b). However, this scalability comes at the cost of ineffective utilization of computing resources and reduced throughput of the system. Moreover, in some cases, processing the entire input data stream would require more than the available computing resources to meet the desired latency/throughput guarantees.

To strike a balance between the two desirable, but contradictory design requirements – low latency and efficient utilization of computing resources – there is a surge of *approximate computing* paradigm that explores a novel design point to resolve this tension. In particular, approximate computing is based on the observation that many data analytics jobs are amenable to an approximate rather than the exact output (Doucet et al. 2000; Natarajan 1995). For such workflows, it is possible to trade the output accuracy by computing over a subset instead of the entire data stream. Since computing over a subset of input requires less time and computing resources, approximate computing can achieve desirable latency and computing resource utilization.

Unfortunately, the advancements in approximate computing are primarily geared towards batch analytics (Agarwal et al. 2013; Srikanth et al. 2016), where the input data remains unchanged during the course of computation. In particular, these systems rely on pre-computing a set of samples on the static database, and take an appropriate sample for the query execution based on the user’s requirements (i.e., query execution budget). Therefore, the state-of-the-art systems cannot be deployed in the context of stream processing, where the new data continuously arrives as an unbounded stream.

As an alternative, one could in principle *repurpose* the available sampling mechanisms in well-known big data processing frameworks such as Apache Spark to build an approximate computing system for stream analytics. In fact, as a starting point for this work, based on the available sampling mechanisms, an approximate computing system is designed and implemented for stream processing in Apache Spark. Unfortunately, Spark’s stratified sampling algorithm suffers from three key limitations for approximate computing. First, Spark’s stratified sampling algorithm operates in a “batch” fashion, i.e., all data items are first collected in a batch as Resilient Distributed Datasets (RDDs) (Zaharia et al. 2012), and thereafter, the actual sampling is carried out on the RDDs. Second, it does not handle the case where the arrival rate of sub-streams changes over time because it requires a pre-defined sampling fraction for each stratum. Lastly, the stratified sampling algorithm implemented in Spark requires synchronization among workers for the expensive join operation, which imposes a significant latency overhead.

To address these limitations, this work designed an *online stratified reservoir sampling algorithm* for stream analytics. Unlike existing Spark-based systems, the algorithm performs the sampling process “on-the-fly” to reduce the latency as well as the overheads associated in the process of forming RDDs. Importantly, the algorithm *generalizes* to two prominent types of stream processing models: (1) batched stream processing employed by Apache Spark Streaming (Foundation 2017b), and (2) pipelined stream processing employed by Apache Flink (Foundation 2017a).

More specifically, the proposed sampling algorithm makes use of two techniques: reservoir sampling and stratified sampling. It performs reservoir sampling for each sub-stream by creating a fixed-size reservoir per stratum. Thereafter, it assigns weights to all strata respecting their arrival rates to preserve the statistical quality of the original data stream. The proposed sampling algorithm naturally adapts to varying arrival rates of sub-streams, and requires no synchronization among workers (see section

“[Design](#)”). Based on the proposed sampling algorithm, STREAMAPPROX – an approximate computing system for stream analytics – is designed.

Overview and Background

This section gives an overview of STREAMAPPROX (section “[System Overview](#)”), its computational model (section “[Computational Model](#)”), and its design assumptions (section “[Design Assumptions](#)”).

System Overview

STREAMAPPROX is designed for real-time stream analytics. In this system, the input data stream usually consists of data items arriving from diverse sources. The data items from each source form a *sub-stream*. The system makes use of a stream aggregator (e.g., Apache Kafka Foundation 2017c) to combine the incoming data items from disjoint sub-streams. STREAMAPPROX then takes this combined stream as the input for data analytics.

STREAMAPPROX facilitate data analytics on the input stream by providing an interface for users to specify the streaming query and its corresponding query budget. The query budget can be in the form of expected latency/throughput guarantees, available computing resources, or the accuracy level of query results.

STREAMAPPROX ensures that the input stream is processed within the specified query budget. To achieve this goal, the system makes use of approximate computing by processing only a subset of data items from the input stream, and produce an approximate output with rigorous error bounds. In particular, STREAMAPPROX uses a parallelizable online sampling technique to select and process a subset of data items, where the sample size can be determined based on the query budget.

Computational Model

The state-of-the-art distributed stream processing systems can be classified in two prominent categories: (i) batched stream processing model,

and (ii) pipelined stream processing model. These systems offer three main advantages: (a) efficient fault tolerance, (b) “exactly-once” semantics, and (c) unified programming model for both batch and stream analytics. *The proposed algorithm for approximate computing is generalizable to both stream processing models, and preserves their advantages.*

Batched stream processing model. In this computational model, an input data stream is divided into small batches using a pre-defined batch interval, and each such batch is processed via a distributed data-parallel job. Apache Spark Streaming (Foundation 2017b) adopted this model to process input data streams.

Pipelined stream processing model. In contrast to the batched stream processing model, the pipelined model streams each data item to the next operator as soon as the item is ready to be processed without forming the whole batch. Thus, this model achieves low latency. Apache Flink (Foundation 2017a) implements this model to provide a truly native stream processing engine.

Note that both stream processing models support the time-based sliding window computation (Bhatotia et al. 2014). The processing window slides over the input stream, whereby the newly incoming data items are added to the window and the old data items are removed from the window. The number of data items within a sliding window may vary in accordance to the arrival rate of data items.

Design Assumptions

STREAMAPPROX is based on the following assumptions. The possible means to address these assumptions are discussed in section “[Discussion](#)”.

1. There exists a virtual cost function which translates a given query budget (such as the expected latency guarantees, or the required accuracy level of query results) into the appropriate sample size.

2. The input stream is stratified based on the source of data items, i.e., the data items from each sub-stream follow the same distribution and are mutually independent. Here, a *stratum* refers to one sub-stream. If multiple sub-streams have the same distribution, they are combined to form a stratum.

Design

In this section, first the STREAMAPPROX’s workflow (section “[System Workflow](#)”) is presented. Then, its sampling mechanism (section “[Online Adaptive Stratified Reservoir Sampling](#)”) and its error estimation mechanism (section “[Error Estimation](#)”) are described (see details in Quoc et al. 2017d,c).

System Workflow

This section shows the workflow of STREAMAPPROX. The system takes the user-specified streaming *query* and the query *budget* as the input. Then it executes the query on the input data stream as a sliding window computation (see section “[Computational Model](#)”).

For each time interval, STREAMAPPROX first derives the sample size (*sampleSize*) using a cost function based on the given query budget. Next, the system performs a proposed sampling algorithm (detailed in section “[Online Adaptive Stratified Reservoir Sampling](#)”) to select the appropriate *sample* in an online fashion. This sampling algorithm further ensures that data items from all sub-streams are fairly selected for the sample, and no single sub-stream is overlooked.

Thereafter, the system executes a data-parallel job to process the user-defined *query* on the selected sample. As the last step, the system performs an error estimation mechanism (as described in section “[Error Estimation](#)”) to compute the error bounds for the approximate query result in the form of $output \pm error\ bound$. The whole process repeats for each time interval as the computation window slides (Bhatotia et al. 2012a).

Online Adaptive Stratified Reservoir Sampling

To realize the real-time stream analytics, a novel sampling technique called Online Adaptive Stratified Reservoir Sampling (OASRS) is proposed. It achieves both stratified and reservoir samplings without their drawbacks. Specifically, OASRS does not overlook any sub-streams regardless of their popularity, does not need to know the statistics of sub-streams before the sampling process, and runs efficiently in real time in a distributed manner.

The high-level idea of OASRS is simple. The algorithm first stratifies the input stream into sub-streams according to their sources. The data items from each sub-stream are assumed to follow the same distribution and are mutually independent. (Here, a *stratum* refers to one sub-stream. If multiple sub-streams have the same distribution, they can be combined to form a stratum.) The algorithm then samples each sub-stream independently, and perform the reservoir sampling for each sub-stream individually. To do so, every time a new sub-stream S_i is encountered, its sample size N_i is determined according to an adaptive cost function considering the specified query budget. For each sub-stream S_i , the algorithm performs the traditional reservoir sampling to select items at random from this sub-stream, and ensures that the total number of selected items from S_i does not exceed its sample size N_i . In addition, the algorithm maintains a counter C_i to measure the number of items received from S_i within the concerned time interval.

Applying reservoir sampling to each sub-stream S_i ensures that algorithm can randomly select at most N_i items from each sub-stream. The selected items from different sub-streams, however, should *not* be treated equally. In particular, for a sub-stream S_i , if $C_i > N_i$ (i.e., the sub-stream S_i has more than N_i items in total during the concerned time interval), the algorithm randomly selects N_i items from this sub-stream and each selected item represents C_i/N_i original items on average; otherwise, if $C_i \leq N_i$, the algorithm selects all the received C_i items so that each selected item only represents itself. As a result, in order to statistically recreate

the original items from the selected items, the algorithm assigns a specific weight W_i to the items selected from each sub-stream S_i :

$$W_i = \begin{cases} C_i/N_i & \text{if } C_i > N_i \\ 1 & \text{if } C_i \leq N_i \end{cases} \quad (1)$$

STREAMAPPROX supports *approximate linear queries* which return an approximate weighted sum of all items received from all sub-streams. Though linear queries are simple, they can be extended to support a large range of statistical learning algorithms (Blum et al. 2005, 2008). It is also worth mentioning that, OASRS not only works for a concerned time interval (e.g., a sliding time window), but also works with unbounded data streams.

Distributed execution. OASRS can run in a distributed fashion naturally as it does not require synchronization. One straightforward approach is to make each sub-stream S_i be handled by a set of w worker nodes. Each worker node samples an equal portion of items from this sub-stream and generates a local reservoir of size no larger than N_i/w . In addition, each worker node maintains a local counter to measure the number of its received items within a concerned time interval for weight calculation. The rest of the design remains the same.

Error Estimation

This section describes how to apply OASRS to randomly sample the input data stream to generate the approximate results for linear queries. Next, a method to estimate the accuracy of approximate results via rigorous error bounds is presented.

Similar to section “[Online Adaptive Stratified Reservoir Sampling](#)”, suppose the input data stream contains X sub-streams $\{S_i\}_{i=1}^X$. STREAMAPPROX computes the approximate sum of all items received from all sub-streams by randomly sampling only Y_i items from each sub-stream S_i . As each sub-stream is sampled independently, the variance of the approximate sum is: $\text{Var}(\text{SUM}) = \sum_{i=1}^X \text{Var}(\text{SUM}_i)$.

Further, as items are randomly selected for a sample within each sub-stream, according to the random sampling theory (Thompson 2012), the variance of the approximate sum can be estimated as:

$$\widehat{\text{Var}}(\text{SUM}) = \sum_{i=1}^X \left(C_i \times (C_i - Y_i) \times \frac{s_i^2}{Y_i} \right) \quad (2)$$

Here, C_i denotes the total number of items from the sub-stream S_i , and s_i denotes the standard deviation of the sub-stream S_i 's sampled items:

$$s_i^2 = \frac{1}{Y_i - 1} \times \sum_{j=1}^{Y_i} (I_{i,j} - \bar{I}_i)^2, \text{ where} \\ \bar{I}_i = \frac{1}{Y_i} \times \sum_{j=1}^{Y_i} I_{i,j} \quad (3)$$

Next, the estimation of the variance of the approximate mean value of all items received from all the X sub-streams is described. This approximate mean value can be computed as:

$$\text{MEAN} = \frac{\text{SUM}}{\sum_{i=1}^X C_i} = \frac{\sum_{i=1}^X (C_i \times \text{MEAN}_i)}{\sum_{i=1}^X C_i} \\ = \sum_{i=1}^X (\omega_i \times \text{MEAN}_i) \quad (4)$$

Here, $\omega_i = \frac{C_i}{\sum_{i=1}^X C_i}$. Then, as each sub-stream is sampled independently, according to the random sampling theory (Thompson 2012), the variance of the approximate mean value can be estimated as:

$$\widehat{\text{Var}}(\text{MEAN}) = \sum_{i=1}^X \text{Var}(\omega_i \times \text{MEAN}_i) \\ = \sum_{i=1}^X \left(\omega_i^2 \times \text{Var}(\text{MEAN}_i) \right) \quad (5) \\ = \sum_{i=1}^X \left(\omega_i^2 \times \frac{s_i^2}{Y_i} \times \frac{C_i - Y_i}{C_i} \right)$$

Above, the estimation of the variances of the approximate sum and the approximate mean of the input data stream has been shown. Similarly, the variance of the approximate results of any linear queries also can be estimated by applying the random sampling theory.

Error bound. According to the “68-95-99.7” rule (Wikipedia 2017), approximate result falls within one, two, and three standard deviations away from the true result with probabilities of 68%, 95%, and 99.7%, respectively, where the standard deviation is the square root of the variance as computed above. This error estimation is critical because it gives a quantitative understanding of the accuracy of the proposed sampling technique.

Discussion

The design of STREAMAPPROX is based on the assumptions mentioned in section “[Design Assumptions](#)”. This section discusses some approaches that could be used to meet the assumptions.

I: Virtual cost function. This work currently assumes that there exists a virtual cost function to translate a user-specified query budget into the sample size. The query budget could be specified as either available computing resources, desired accuracy, or latency.

For instance, with an accuracy budget, the sample size for each sub-stream can be determined based on a desired width of the confidence interval using Eq. (5) and the “68-95-99.7” rule. With a desired latency budget, users can specify it by defining the window time interval or the slide interval for the computations over the input data stream. It becomes a bit more challenging to specify a budget for resource utilization. Nevertheless, there are two existing techniques that could be used to implement such a cost function to achieve the desired resource target: (a) virtual data center (Angel et al. 2014), and (b) resource prediction model (Wieder et al. 2012) for latency requirements.

Pulsar (Angel et al. 2014) proposes an abstraction of a virtual data center (VDC) to provide performance guarantees to tenants in the cloud. In particular, Pulsar makes use of a virtual cost function to translate the cost of a request processing into the required computational resources using a multi-resource token algorithm. The cost function could be adapted for STREAMAPPROX as follows: a data item in the input stream is considered as a request and the “amount of resources” required to process it as the cost in tokens. Also, the given resource budget is converted in the form of tokens, using the pre-advertised cost model per resource. This allows computing the sample size that can be processed within the given resource budget.

For any given latency requirement, resource prediction model (Wieder et al. 2010a,b, 2012) could be employed. In particular, the prediction model could be built by analyzing the diurnal patterns in resource usage (Charles et al. 2012) to predict the future resource requirement for the given latency budget. This resource requirement can then be mapped to the desired sample size based on the same approach as described above.

II: Stratified sampling. This work currently assume that the input stream is already stratified based on the source of data items, i.e., the data items within each stratum follow the same distribution – it does not have to be a normal distribution. This assumption ensures that the error estimation mechanism still holds correct since STREAMAPPROX applies the Central Limit Theorem. For example, consider an IoT use-case which analyzes data streams from sensors to measure the temperature of a city. The data stream from each individual sensor follows the same distribution since it measures the temperature at the same location in the city. Therefore, a straightforward way to stratify the input data streams is to consider each sensor’s data stream as a stratum (sub-stream). In more complex cases where STREAMAPPROX cannot classify strata based on the sources, the system needs a pre-processing step to stratify the input data stream. This stratification problem is orthogonal to this work,

nevertheless for completeness, two proposals for the stratification of evolving streams, bootstrap (Dziuda 2010) and semi-supervised learning (Masud et al. 2012), are discussed in this section.

Bootstrap (Dziuda 2010) is a well-studied non-parametric sampling technique in statistics for the estimation of distribution for a given population. In particular, the bootstrap technique randomly selects “bootstrap samples” with replacement to estimate the unknown parameters of a population, for instance, by averaging the bootstrap samples. A bootstrap-based estimator can be employed for the stratification of incoming sub-streams. Alternatively, a semi-supervised algorithm (Masud et al. 2012) could be used to stratify a data stream. The advantage of this algorithm is that it can work with both labeled and unlabeled streams to train a classification model.

Related Work

Over the last two decades, the databases community has proposed various approximation techniques based on sampling (Al-Kateb and Lee 2010; Garofalakis and Gibbon 2001), online aggregation (Hellerstein et al. 1997), and sketches (Cormode et al. 2012). These techniques make different trade-offs w.r.t. the output quality, supported queries, and workload. However, the early work in approximate computing was mainly geared towards the centralized database architecture.

Recently, sampling-based approaches have been successfully adopted for distributed data analytics (Agarwal et al. 2013; Srikanth et al. 2016; Krishnan et al. 2016; Quoc et al. 2017b,a). In particular, BlinkDB (Agarwal et al. 2013) proposes an approximate distributed query processing engine that uses stratified sampling (Al-Kateb and Lee 2010) to support ad-hoc queries with error and response time constraints. Like BlinkDB, Quickr (Srikanth et al. 2016) also supports complex ad-hoc queries in big-data clusters. Quickr deploys distributed sampling operators to reduce execution costs of

parallelized queries. In particular, Quickr first injects sampling operators into the query plan; thereafter, it searches for an optimal query plan among sampled query plans to execute input queries. However, these “big data” systems target batch processing and cannot provide required low-latency guarantees for stream analytics.

IncApprox (Krishnan et al. 2016) is a data analytics system that combines two computing paradigms together, namely, approximate and incremental computations (Bhatotia et al. 2011a,b, 2012b) for stream analytics. The system is based on an online “biased sampling” algorithm that uses self-adjusting computation (Bhatotia 2015; Bhatotia et al. 2015) to produce incrementally updated approximate output. Lastly, PrivApprox (Quoc et al. 2017a,b) supports privacy-preserving data analytics using a combination of randomized response and approximate computation. By contrast, STREAMAPPROX supports low-latency in stream processing by employing the proposed “online” sampling algorithm solely for approximate computing, while avoiding the limitations of existing sampling algorithms.

Conclusion

This paper presents STREAMAPPROX, a stream analytics system for approximate computing. STREAMAPPROX allows users to make a systematic trade-off between the output accuracy and the computation efficiency. To achieve this goal, STREAMAPPROX employs an online stratified reservoir sampling algorithm which ensures the statistical quality of the sample selected from the input data stream. The proposed sampling algorithm is generalizable to two prominent types of stream processing models: batched and pipelined stream processing models.

Cross-References

► Incremental Approximate Computing

References

- Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the ACM European conference on computer systems (EuroSys)
- Al-Kateeb M, Lee BS (2010) Stratified reservoir sampling over heterogeneous data streams. In: Proceedings of the 22nd international conference on scientific and statistical database management (SSDBM)
- Angel S, Ballani H, Karagiannis T, O’Shea G, Thereska E (2014) End-to-end performance isolation through virtual datacenters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Bhatotia P (2015) Incremental parallel and distributed systems. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS)
- Bhatotia P, Wieder A, Akkus IE, Rodrigues R, Acar UA (2011a) Large-scale incremental data processing with change propagation. In: Proceedings of the conference on hot topics in cloud computing (HotCloud)
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquini R (2011b) Incoop: MapReduce for incremental computations. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Bhatotia P, Dischinger M, Rodrigues R, Acar UA (2012a) Slider: incremental sliding-window computations for large-scale data analysis. Technical report MPI-SWS-2012-004, MPI-SWS. <http://www.mpi-sws.org/tr/2012-004.pdf>
- Bhatotia P, Rodrigues R, Verma A (2012b) Shredder: GPU-accelerated incremental storage and computation. In: Proceedings of USENIX conference on file and storage technologies (FAST)
- Bhatotia P, Acar UA, Junqueira FP, Rodrigues R (2014) Slider: incremental sliding window analytics. In: Proceedings of the 15th international middleware conference (Middleware)
- Bhatotia P, Fonseca P, Acar UA, Brandenburg B, Rodrigues R (2015) iThreads: a threading library for parallel incremental computation. In: Proceedings of the 20th international conference on architectural support for programming languages and operating systems (ASPLOS)
- Blum A, Dwork C, McSherry F, Nissim K (2005) Practical privacy: the sułq framework. In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS)
- Blum A, Ligett K, Roth A (2008) A learning theory approach to non-interactive database privacy. In: Proceedings of the fortieth annual ACM symposium on theory of computing (STOC)
- Charles R, Alexey T, Gregory G, Randy HK, Michael K (2012) Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical report

- Cormode G, Garofalakis M, Haas PJ, Jermaine C (2012) Synopses for massive data: samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*. Now, Boston
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Doucet A, Godsill S, Andrieu C (2000) On sequential monte carlo sampling methods for bayesian filtering. *Stat Comput* 10:197–208
- Dziuda DM (2010) Data mining for genomics and proteomics: analysis of gene and protein expression data. Wiley, Hoboken
- Foundation AS (2017a) Apache flink. <https://flink.apache.org>
- Foundation AS (2017b) Apache spark streaming. <https://spark.apache.org/streaming>
- Foundation AS (2017c) Kafka – a high-throughput distributed messaging system. <https://kafka.apache.org>
- Garofalakis MN, Gibbon PB (2001) Approximate query processing: taming the terabytes. In: Proceedings of the international conference on very large data bases (VLDB)
- Hellerstein JM, Haas PJ, Wang HJ (1997) Online aggregation. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Krishnan DR, Quoc DL, Bhatotia P, Fetzer C, Rodrigues R (2016) IncApprox: a data analytics system for incremental approximate computing. In: Proceedings of the 25th international conference on World Wide Web (WWW)
- Masud MM, Woolam C, Gao J, Khan L, Han J, Hamlen KW, Oza NC (2012) Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowl Inf Syst* 33:213–244
- Murray DG, McSherry F, Isaacs R, Isard M, Barham P, Abadi M (2013) Naiad: a timely dataflow system. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles (SOSP)
- Natarajan S (1995) Imprecise and approximate computation. Kluwer Academic Publishers, Boston
- Quoc DL, Martin A, Fetzer C (2013) Scalable and real-time deep packet inspection. In: Proceedings of the 2013 IEEE/ACM 6th international conference on utility and cloud computing (UCC)
- Quoc DL, Yazdanov L, Fetzer C (2014) Dolen: user-side multi-cloud application monitoring. In: International conference on future internet of things and cloud (FI-CLOUD)
- Quoc DL, D'Alessandro V, Park B, Romano L, Fetzer C (2015a) Scalable network traffic classification using distributed support vector machines. In: Proceedings of the 2015 IEEE 8th international conference on cloud computing (CLOUD)
- Quoc DL, Fetzer C, Felber P, Étienne Rivière, Schiavoni V, Sutra P (2015b) Unicrawl: a practical geographically distributed web crawler. In: Proceedings of the 2015 IEEE 8th international conference on cloud computing (CLOUD)
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017a) Privacy preserving stream analytics: the marriage of randomized response and approximate computing. <https://arxiv.org/abs/1701.05403>
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017b) PrivApprox: privacy-preserving stream analytics. In: Proceedings of the 2017 USENIX conference on USENIX annual technical conference (USENIX ATC)
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017c) Approximate stream analytics in apache flink and apache spark streaming. CoRR, abs/1709.02946
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017d) StreamApprox: approximate computing for stream analytics. In: Proceedings of the international middleware conference (middleware)
- Srikanth K, Anil S, Aleksandar V, Matthaios O, Robert G, Surajit C, Ding B (2016) Quickr: lazily approximating complex ad-hoc queries in big data clusters. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Thompson SK (2012) Sampling. Wiley series in probability and statistics. The Australasian Institute of Mining and Metallurgy, Carlton
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010a) Brief announcement: modelling mapreduce for optimal execution in the cloud. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing (PODC)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010b) Conductor: orchestrating the clouds. In: Proceedings of the 4th international workshop on large scale distributed systems and middleware (LADIS)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2012) Orchestrating the deployment of computations in the cloud with conductor. In: Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI)
- Wikipedia (2017) 68-95-99.7 Rule
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation (NSDI)
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles (SOSP)

Approximate Reasoning

► Automated Reasoning

Approximated Neighborhood Function

- ▶ Degrees of Separation and Diameter in Large Graphs

Architectures

Erik G. Hoel
Environmental Systems Research Institute,
Redlands, CA, USA

Synonyms

[Apache Hadoop](#); [Lambda architecture](#); [Location analytic architecture](#)

Definitions

Spatial big data is a spatio-temporal data that is too large or requires data-intensive computation that is too demanding for traditional computing architectures. *Stream processing* in this context is the processing of spatio-temporal data in motion. The data is observational; it is produced by sensors – moving or otherwise. Computations on the data are made as the data is produced or received. A *distributed processing cluster* is a networked collection of computers that communicate and process data in a coordinated manner. Computers in the cluster are coordinated to solve a common problem. A *lambda architecture* is a scalable, fault-tolerant data-processing architecture that is designed to handle large quantities of data by exploiting both stream and batch processing methods. *Data partitioning* involves physically dividing a dataset into separate data stores on a distributed processing cluster. This is done to achieve improved scalability, performance, availability, and fault-tolerance. *Distributed file systems*, in the context of big data architectures, are similar to traditional distributed file systems but are intended to persist large datasets on com-

modity hardware in a fault-tolerant manner with simple coherency models. The *MapReduce programming model* is intended for large-scale distributed data processing and is based upon simple concepts involving iterating over data, performing a computation on key/value pairs, grouping the intermediary values by key, iterating over the resulting groups, and reducing each group to provide a result. A *GPU-accelerated distributed processing framework* is an extension to a traditional distributed processing framework that supports offloading tasks to GPUs for further acceleration.

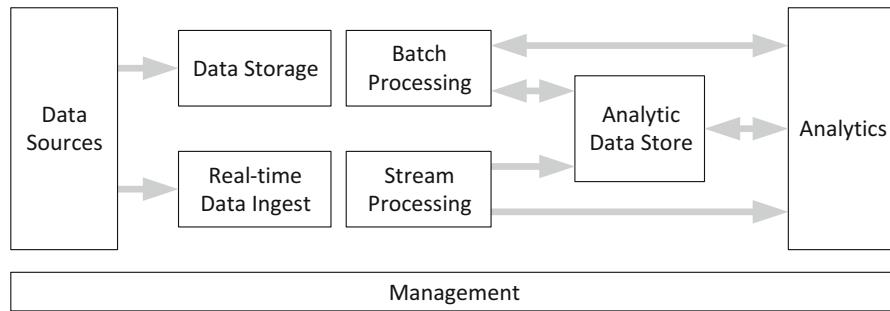
Overview

Spatial big data architectures are intended to address requirements for spatio-temporal data that is too large or computationally demanding for traditional computing architectures (Shekhar et al. 2012). This includes operations such as real-time data ingest, stream processing, batch processing, storage, and spatio-temporal analytical processing.

Spatial big data offers additional challenges beyond what is commonly faced in the big data space. This includes advanced indexing, querying, analytical processing, visualization, and machine learning. Examples of spatial big data include moving vehicles (peer-to-peer ridesharing, delivery vehicles, ships, airplanes, etc.), stationary sensor data (e.g., SCADA, AMI), cell phones (call detail records), IoT devices, as well as spatially enabled content from social media (Fig. 1).

Spatial big data architectures are used in a variety of workflows:

- Real-time processing of observational data (data in motion). This commonly involves monitoring and tracking dynamic assets in real-time; this can include vehicles, aircraft, and vessels, as well as stationary assets such as weather and environmental monitoring sensors.
- Batch processing of persisted spatio-temporal data (data at rest). Workflows with data at rest incorporate tabular and spatial processing



Architectures, Fig. 1 Generic big data architecture

(e.g., summarizing data, analyzing patterns, and proximity analysis), geoenrichment and geoenablement (adding spatial capabilities to non-spatial data, adding information from contextual spatial data collections), and machine learning and predictive analytics (clustering, classification, and prediction).

Architectural Requirements

When designing and developing scalable software systems that can address the high-level workflows encountered in spatial big data systems, a number of basic requirements must be identified. It is important to note that most of these also generally apply to traditional non-spatial big data systems (Mysore et al. 2013; Klein et al. 2016; Sena et al. 2017):

Scalable

Spatial big data systems must be scalable. Scalability encompasses being able to increase and support different amounts of data, processing them, and allocating computational resources without impacting costs or efficiency. To meet this requirement, it is required to distribute data sets and their processing across multiple computing and storage nodes.

High Performant

Spatial big data systems must be able to process large streams of data in a short period of time, thus returning the results to users as efficiently as possible. In addition, the system should support computation intensive spatio-temporal analytics.

Real-time

Big data systems must be able to manage the continuous flow of data and its processing in real time, facilitating decision-making.

Consistent

Big data systems must support data consistency, heterogeneity, and exploitation. Different data formats must be also managed to represent useful information for the system.

Secure

Big data systems must ensure security in the data and its manipulation in the architecture, supporting integrity of information, exchanging data, multilevel policy-driven, access control, and prevent unauthorized access.

Available

Big data systems must ensure high data availability, through data replication horizontal scaling (i.e., distribute a data set over clusters of computers and storage nodes). The system must allow replication and handle hardware failures.

Interoperable

Big data systems must be transparently intercommunicated to allow exchanging information between machines and processes, interfaces, and people.

Key Research Findings

Spatial big data architectures have existed since the early 2000s with some of the original implementations at Google and Microsoft. Some of the key research issues related to spatial big

data architectures include data storage (repositories, distributes storage, NoSQL databases), distributed spatio-temporal analytic processing, stream processing, scalable cloud computing, and GPU-enabled distributed processing frameworks.

Data Storage

Parallel Database Systems

Big data repositories have existed in many forms, frequently built by corporations and governmental agencies with special requirements. Beginning in the 1990s, commercial vendors began offering parallel database management systems to address the needs of big data (DeWitt and Gray 1992). Parallel database systems are classified as being shared memory (processing elements sharing memory), shared disk (processing elements do not share memory, but do share disk storage), or shared nothing (neither memory nor disk storage is shared between processing elements). Significant efforts included those by Teradata, IBM, Digital, Microsoft, Oracle, Google, and Amazon. Additionally, beginning in the 1980s, there were numerous research systems that contributed to these efforts (e.g., GAMMA and Bubba; DeWitt et al. 1986, Alexander and Copeland 1988).

Distributed File Stores

Hadoop Distributed File System (HDFS) is a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster (Shvachko et al. 2010). It was inspired by the Google File System (GFS; Ghemawat et al. 2003). HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence theoretically does not require redundant array of independent disks (RAID) storage on hosts (but to increase input-output (I/O) performance, RAID configurations may be employed). With the default replication value of three, data is stored on three nodes: two on the same rack and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

NoSQL Databases

A NoSQL (originally referencing “non-SQL” or “non-relational”) database is a mechanism for the storage and retrieval of data that is modeled differently from standard relational databases (NoSQL 2009; Pavlo and Aslett 2016). NoSQL databases are often considered next generation databases; they are intended to address weaknesses of traditional relational databases such as being readily distributable, simpler in design, open-source, and horizontally scalable (often problematic for relational databases). Many databases supporting these characteristics originated in the late 1960s; the “NoSQL” description was employed beginning in the late 1990s with the requirements imposed by companies such as Facebook, Google, and Amazon. NoSQL databases are commonly used with big data applications. NoSQL systems are also sometimes called “Not only SQL” to emphasize that they may support SQL-like query languages.

In order to achieve increased performance and scalability, NoSQL databases commonly used data structures (e.g., key-value, columnar, document, or graph) that are different from those used in relational databases. NoSQL databases vary in terms of applicability to particular problem domains. NoSQL databases are often classified by their primal data structures; examples include:

- Key-value: Apache Ignite, Couchbase, Dynamo, Oracle NoSQL Database, Redis, Riak
- Columnar: Accumulo, Cassandra, Druid, HBase, Vertica
- Document: Apache CouchDB, Cosmos DB, IBM Domino, MarkLogic, MongoDB
- Graph: AllegroGraph, Apache Giraph, MarkLogic, Neo4J
- Multi-model: Apache Ignite, Couchbase, MarkLogic

Spatial Batch Processing

Spatio-temporal analysis in a batch context involves a very wide scope of functionality. In academia, much of the research has focused on the spatial join (or spatio-temporal join) func-

tion. In commercial systems, spatial analysis also includes summarizing data, incident and similar location detection, proximity analysis, pattern analysis, and data management (import, export, cleansing, etc.).

Spatial Joins

Spatial joins have been widely studied in both the standard sequential environment (Jacox and Samet 2007), as well as in the parallel (Brinkhoff et al. 1996) and distributed environments (Abel et al. 1995). For over 20 years, algorithms have been developed to take advantage of parallel and distributed processing architectures and software frameworks. The recent resurgence in interest in spatial join processing is the results of new-found interest in distributed, fault-tolerant, computing frameworks such as Apache Hadoop, as well as the explosion in observational and IoT data.

With distributed processing architectures, there are two principal approaches that are employed when performing spatial joins. The first, termed a broadcast (or mapside) spatial join, is designed for joining a large dataset with another small dataset (e.g., political boundaries). The large dataset is partitioned across the processing nodes and the complete small dataset is broadcast to each of the nodes. This allows significant optimization opportunities. The second approach, termed a partitioned (or reduce side) spatial join, is a more general technique that is used when joining two large datasets. Partitioned joins use a divide-and-conquer approach (Aji et al. 2013). The two large datasets are divided into small pieces via a spatial decomposition, and each small piece is processed independently.

SJMR (Spatial Join with MapReduce) introduced the first distributed spatial join on Hadoop using the MapReduce programming model (Dean and Ghemawat 2008; Zhang et al. 2009). Spatial-Hadoop (Eldawy and Mokbel 2015) optimized SJMR with a persistent spatial index (it supports grid files, R – trees, and R + trees) that is precomputed. Hadoop-GIS (Aji et al. 2013), which is utilized in medical pathology imaging, features both 2D and 3D spatial join. GIS Tools

for Hadoop (Whitman et al. 2014) is an open source library that implements range and distance queries and k-NN. It also supports a distributed PMR quadtree-based spatial index. GeoSpark (Yu et al. 2015) is a framework for performing spatial joins, range, and k-NN queries. The framework supports quadtree and R-tree indexing of the source data. Magellan (Sriharsha 2015) is an open source library for geospatial analytics that uses Spark (Zaharia et al. 2010). It supports a broadcast join and a reduce-side optimized join and is integrated with Spark SQL for a traditional, SQL user experience. SpatialSpark (You et al. 2015) supports both a broadcast spatial join and a partitioned spatial join on Spark. The partitioning is supported using either fixed-grid, binary space partition or a sort-tile approach. STARK (Hagedorn et al. 2017) is a Spark-based framework that supports spatial joins, k-NN, and range queries on both spatial and spatio-temporal data. STARK supports three temporal operators: contains, containedBy, and intersects) and also supports the DBSCAN density-based spatial clusterer (Ester et al. 1996). MSJS (multi-way spatial join algorithm with Spark (Du et al. 2017)) addresses the problem of performing multi-way spatial joins using the common technique of cascading sequences of pairwise spatial joins. Simba (Xie et al. 2016) offers range, distance (circle range), and k-NN queries as well as distance and k-NN joins. Two-level indexing, global and local, is employed, similar to the various indexing work on Hadoop MapReduce. Location-Spark (Tang et al. 2016) supports range query, k-NN, spatial join, and k-NN join. Location-Spark uses global and local indices – Grid, R-tree, Quadtree, and IR-tree. GeoMesa is an open-source, distributed, spatio-temporal index built on top of Bigtable-style databases (Chang et al. 2008) using an implementation of the Geohash algorithm implemented in Scala (Hughes et al. 2015). The Esri GeoAnalytics Server (Whitman et al. 2017) supports many types of spatial analysis in a distributed environment (leveraging the Spark framework). It provides functionality for summarizing data (e.g., aggregation, spatio-temporal join, polygon overlay), incident and similar location detection, proximity

analysis, and pattern analysis (hot spot analysis, NetCDF generation).

MapReduce Programming Model

MapReduce is a programming model and an associated implementation for processing big data sets with a parallel, distributed algorithm (Sakr et al. 2013). A MapReduce program is composed of a map procedure (or method), which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a reduce method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). A MapReduce framework manages the processing by marshalling the distributed cluster nodes, running the various tasks and algorithms in parallel, managing communications and data transfers between cluster nodes, while supporting fault tolerance and redundancy.

The MapReduce model is inspired by the map and reduces functions commonly used in functional programming (note that their purpose in the MapReduce framework is not the same as in their original forms). The key contributions of the MapReduce model are not the actual map and reduce functions that resemble the Message Passing Interface (MPI) standard's reduce and scatter operations. The major contributions of the MapReduce model are the scalability and fault-tolerance that is supported through optimization of the execution engine. A single-threaded implementation of MapReduce is commonly slower than a traditional (non-MapReduce) implementation; gains are typically realized with multi-node or multi-threaded implementations.

MapReduce libraries have been written in many programming languages, with different levels of optimization. The most popular open-source implementation is found in Apache Hadoop.

Stream Processing

Stream processing is a computer programming paradigm, equivalent to dataflow programming, event stream processing, and reactive programming that allows some applications to more easily exploit a limited form of parallel processing

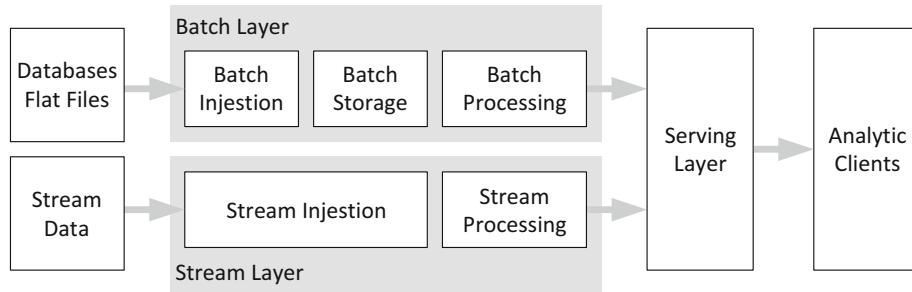
(Gedik et al. 2008). Such applications can use multiple computational units, such as the floating point unit on a graphics processing unit or field-programmable gate arrays (FPGAs), without explicitly managing allocation, synchronization, or communication among those units.

The stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed. Given a sequence of data (a stream), a series of operations (kernel functions) is applied to each element in the stream. Kernel functions are usually pipelined, and optimal local on-chip memory reuse is attempted, in order to minimize the loss in bandwidth, accredited to external memory interaction. Uniform streaming, where one kernel function is applied to all elements in the stream, is typical. Since the kernel and stream abstractions expose data dependencies, compiler tools can fully automate and optimize on-chip management tasks.

Lambda Architecture

A Lambda Architecture is an architecture that is intended to process large volumes of data by incorporating both batch and real-time processing techniques (Marz and Warren 2015). This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. The two view outputs may be joined before presentation. In addition, historic analysis is used to tune the real-time analytical processing as well as building models for prediction (machine learning). The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latencies of map-reduce (Fig. 2).

The Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record. It is intended for ingesting and processing timestamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data.



Architectures, Fig. 2 Generic lambda architecture

GPU-Accelerated Distributed Frameworks

Distributed processing frameworks such as Spark (which support in-memory processing) have been extended and enhanced with the incorporation of GPUs for key computationally intensive operations (e.g., machine learning and graph theoretic algorithms, Prasad et al. 2015). Researchers have observed that a few Spark nodes with GPUs can outperform a much larger cluster of non-GPU nodes (Grossman and Sarkar 2016; Hasaan and Elghandour 2016; Yuan et al. 2016; Hong et al. 2017). The main bottlenecks when incorporating GPUs in hybrid architectures often involve data communication, memory and resource management, and differences in programming models. Different approaches to solving these problems have employed GPU-wrapper APIs (e.g., PyCUDA), hybrid RDDs (resilient distributed datasets) where the RDD is stored in the CPU, generating native GPU code from high-level source code written for the distributed framework (e.g., Scala, Java, or Python code with Spark) or native GPU RDDs where data is processed and stored in the GPU device memory (Fig. 3).

Examples of Application

The application of technologies related to spatial big data architectures is broad given the rapidly growing interest in spatial data that has emerged during the twenty-first century. Notable among this family of technologies in terms of significance and application include distributed processing frameworks, geo-spatial stream pro-

cessing, and the numerous implementations of platform as a service (PaaS).

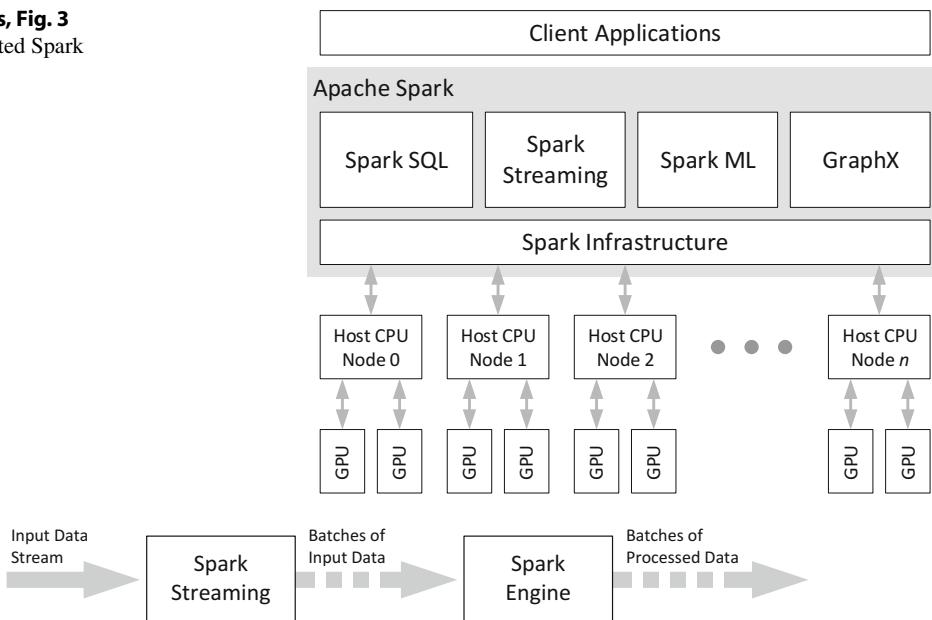
Apache Hadoop

Apache Hadoop (Apache 2006) is an open-source software framework and associated utilities that facilitate using a network of commodity computers to solve problems involving large amounts of data and computation. Inspired by the seminal work at Google on MapReduce and the Google File System (GFS), Hadoop provides a software framework for both the distributed storage and processing of big data using the MapReduce programming model.

Similar to the efforts at Google, Hadoop was designed for computer clusters built from commodity hardware (still the common usage pattern). Hadoop has also been employed on large clusters of higher-end hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), a resource manager, a collection of utilities, and a processing framework that is an implementation of the MapReduce programming model that runs against large clusters of machines. HDFS splits very large files (including those of size gigabytes and larger) into blocks that are distributed across multiple nodes in a cluster. Reliability is achieved by replicating the blocks across multiple nodes (with a default replication factor of 3). Hadoop distributes packaged code into nodes to process the data in parallel.

Architectures, Fig. 3
GPU-accelerated Spark framework



Architectures, Fig. 4 Spark streaming micro-batch architecture

This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Hadoop has been deployed in traditional data-centers as well as in the cloud. The cloud allows organizations to deploy Hadoop without the need to acquire hardware or specific setup expertise. Vendors who currently have an offering for the cloud that incorporate Hadoop include Microsoft, Amazon, IBM, Google, and Oracle. Most of the Fortune 50 companies currently deploy Hadoop clusters.

Spark Streaming

Spark Streaming is an extension to Spark API that supports scalable, high-throughput, fault-tolerant, stream processing of real-time data streams (Garillot and Maas 2018). Data can be ingested from many sources (e.g., Kafka, Flume, or TCP sockets) and can be processed using

temporally aware algorithms expressed with high-level functions like map, reduce, join, and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. Spark's machine learning (Spark ML) and graph processing (GraphX) algorithms can be applied to these data streams.

Internally, Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches (Fig. 4).

Spark Streaming provides a high-level abstraction called *discretized stream* or *DStream*, which represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

Big Data as a Service

Platform as a Service (PaaS) is a category of cloud computing services that provides a platform allowing customers to run and

manage applications without the complexity of building and maintaining the infrastructure usually associated with developing and launching an application (Chang et al. 2010). PaaS is commonly delivered in one of three ways:

- As a public cloud service from a provider
- As a private service (software or appliance) inside the firewall
- As software deployed on a public infrastructure as a service

Big Data as a Service (BDaaS) is a new concept that combines Software as a Service (SaaS), Platform as a Service (PaaS), and Data as a Service (DaaS) in order to address the requirements of working with massively large data sets. BDaaS offerings commonly incorporate the Hadoop stack (e.g., HDFS, Hive, MapReduce, Pig, Storm, and Spark), NoSQL data stores, and stream processing capabilities.

Microsoft Azure is a cloud computing service utilizing Microsoft-managed data centers that supports both software as a service (SaaS) and platform as a service (PaaS). It provides data storage capabilities including Cosmos DB (a NoSQL database), the Azure Data Lake, and SQL Server-based databases. Azure supports a scalable event processing engine and a machine learning service that supports predictive analytics and data science applications.

The Google Cloud is a Paas offering that supports big data with data warehousing, batch and stream processing, data exploration, and support for the Hadoop/Spark framework. Key components include BigQuery, a managed data warehouse supporting analytics at scale, Cloud Dataflow, which supports both stream and batch processing, and Cloud Dataproc, a framework for running Apache MapReduce and Spark processes.

Amazon AWS, though commonly considered an Infrastructure as a Service (IaaS) where the user is responsible for configuration, AWS also provides PaaS functionality. Amazon supports Elastic MapReduce (EMR) that works in conjunction with EC2 (Elastic Compute Cloud) and

S3 (Simple Storage Service). Data storage is provided through DynamoDB (NoSQL), Redshift (columnar), and RDS (relational data store). Machine learning and real-time data processing infrastructures are also supported.

Other significant examples of BDaaS providers include the IBM Cloud and the Oracle Data Cloud. Big data Infrastructure as a Service (IaaS) offerings (that work with other clouds such as AWS, Azure, and Oracle) are available from Hortonworks, Cloudera, Esri, and Databricks.

Future Directions for Research

Despite the significant advancements that have been made over the past decade on key topics related to spatial big data architectures, much further research is necessary in order to further democratize the capabilities and application to broader problem domains. Some of the more significant areas needing attention include:

- Spatio-temporally enabling distributed and NoSQL databases such as Accumulo, Cassandra, HBase, Dynamo, and Elasticsearch. This involves not only supporting spatial types but also incorporating rich collections of topological, spatial, and temporal operators.
- Spatio-temporal analytics is another area requiring attention. Much research to date has focused on supporting spatial (or spatio-temporal) joins on distributed frameworks such as MapReduce or Spark. While beneficial, spatio-temporal analytics is a far richer domain that also includes geostatistics (e.g., kriging), spatial statistics, proximity analysis, and pattern analysis.
- Spatially enabling machine learning algorithms that run in a distributed cluster (e.g., extending Spark ML or Scikit-learn (Pedregosa et al. 2011)) is another significant research area given the growing interest and importance of machine learning, predictive analytics, and deep learning. To date, research

- has primarily focused on density-based clustering algorithms such as DBSCAN, HDBSCAN (McInnes and Healy 2017), and OPTICS.
- Recently, much attention has been paid to incorporating GPU processing capabilities into distributed processing frameworks such as Spark. While some basic spatial capabilities can currently be supported (e.g., aggregation and visualization of point data), much work needs to be done to further streamline and optimize the integration of GPU processors and extend the native spatio-temporal capabilities.

Cross-References

- ▶ [Big Data Architectures](#)
- ▶ [Real-Time Big Spatial Data Processing](#)
- ▶ [Streaming Big Spatial Data](#)

References

- Abel DJ, Ooi BC, Tan K-L, Power R, Yu JX (1995) Spatial join strategies in distributed spatial DBMS. In: Advances in spatial databases – 4th international symposium, SSD'95. Lecture notes in computer science, vol 1619. Springer, Portland, pp 348–367
- Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J (2013) Hadoop-GIS: a high performance spatial data warehousing system over mapreduce. Proc VLDB Endow 6(11):1009–1020
- Alexander W, Copeland G (1988) Process and dataflow control in distributed data-intensive systems. In: Proceedings of the 1988 ACM SIGMOD international conference on management of data (SIGMOD '88), pp 90–98. <https://doi.org/10.1145/50202.50212>
- Apache (2006) Welcome to Apache Hadoop!. <http://hadoop.apache.org>. Accessed 26 Mar 2018
- Brinkhoff T, Kriegel HP, Seeger B (1996) Parallel processing of spatial joins using r-trees. In: Proceedings of the 12th international conference on data engineering, New Orleans, Louisiana, pp 258–265
- Chang F, Dean J, Ghemawat S, Hsieh W, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2). <https://doi.org/10.1145/1365815.1365816>
- Chang WY, Abu-Amara H, Sanford JF (2010) Transforming Enterprise Cloud Services. Springer, London, pp 55–56
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113. <https://doi.org/10.1145/1327452.1327492>
- DeWitt D, Gray J (1992) Parallel database systems: the future of high performance database systems. Commun ACM 35(6). <https://doi.org/10.1145/129888.129894>
- DeWitt DJ, Gerber RH, Graefe G, Heytens ML, Kumar KB, Muralikrishna M (1986) GAMMA – a high performance dataflow database machine. In: Proceedings of the 12th international conference on very large data bases (VLDB '86), Kyoto, Japan, pp 228–237
- Du Z, Zhao X, Ye X, Zhou J, Zhang F, Liu R (2017) An effective high-performance multiway spatial join algorithm with spark. ISPRS Int J Geo-Information 6(4):96
- Eldawy A, Mokbel MF (2015) SpatialHadoop: a mapreduce framework for spatial data. In: IEEE 31st international conference on data engineering (ICDE), Seoul, South Korea, pp 1352–1363
- Ester M, Kriegel HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd international conference on knowledge discovery and data mining (KDD-96), Portland, Oregon, pp 226–231
- Garillot F, Maas G (2018) Stream processing with apache spark: best practices for scaling and optimizing Apache spark. O'Reilly Media, Sebastopol. <http://shop.oreilly.com/product/0636920047568.do>
- Gedik B, Andrade H, Wu K-L, Yu PS, Doo M (2008) SPADE: the system's declarative stream processing engine. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD '08), pp 1123–1134. <https://doi.org/10.1145/1376616.1376729>
- Ghemawat S, Gobioff H, Leung S (2003) The Google file system. In: Proceedings of the 19th ACM symposium on operating systems principles, Oct 2003, pp 29–43. <https://doi.org/10.1145/945445.945450>
- Grossman M, Sarkar V (2016) SWAT: a programmable, in-memory, distributed, high-performance computing platform. In: Proceedings of the 25th ACM international symposium on high-performance parallel and distributed computing (HPDC '16). ACM, New York, pp 81–92. <https://doi.org/10.1145/2907294.2907307>
- Hagedorn S, Götz P, Sattler KU (2017) The STARK framework for spatio-temporal data analytics on spark. In: Proceedings of the 17th conference on database systems for business, technology, and the web (BTW 2017), Stuttgart
- Hassaan M, Elghandour I (2016) A real-time big data analysis framework on a CPU/GPU heterogeneous cluster: a meteorological application case study. In: Proceedings of the 3rd IEEE/ACM international conference on big data computing, applications and tech-

- nologies (BDCAT '16). ACM, New York, pp 168–177. <https://doi.org/10.1145/3006299.3006304>
- Hong S, Choi W, Jeong W-K (2017) GPU in-memory processing using spark for iterative computation. In: Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CC-Grid '17), pp 31–41. <https://doi.org/10.1109/CCGRID.2017.41>
- Hughes JN, Annex A, Eichelberger CN, Fox A, Hulbert A, Ronquest M (2015) Geomesa: a distributed architecture for spatio-temporal fusion. In: Proceedings of SPIE defense and security. <https://doi.org/10.11117/12.2177233>
- Jacox EH, Samet H (2007) Spatial join techniques. *ACM Trans Database Syst* 32(1):7
- Klein J, Buglak R, Blockow D, Wuttke T, Cooper B (2016) A reference architecture for big data systems in the national security domain. In: Proceedings of the 2nd international workshop on BIG data software engineering (BIGDSE '16). <https://doi.org/10.1145/2896825.2896834>
- Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems, 1st edn. Manning Publications, Greenwich
- McInnes L, Healy J (2017) Accelerated hierarchical density based clustering. In: IEEE international conference on data mining workshops (ICDMW), New Orleans, Louisiana, pp 33–42
- Mysore D, Khupat S, Jain S (2013) Big data architecture and patterns. IBM, White Paper, 2013. <http://www.ibm.com/developerworks/library/bdarchpatterns1>. Accessed 26 Mar 2018
- NoSQL (2009) NoSQL definition. <http://nosql-database.org>. Accessed 26 Mar 2018
- Pavlo A, Aslett M (2016) What's really new with NewSQL? *SIGMOD Rec* 45(2):45–55. <https://doi.org/10.1145/3003665.3003674>
- Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Perrot M, Duchesnay É (2011) Scikit-learn: machine learning in Python. *J Mach Learn Res* 12:2825–2830
- Prasad S, McDermott M, Puri S, Shah D, Aghajarian D, Shekhar S, Zhou X (2015) A vision for GPU-accelerated parallel computation on geo-spatial datasets. *SIGSPATIAL Spec* 6(3):19–26. <https://doi.org/10.1145/2766196.2766200>
- Sakr S, Liu A, Fayoumi AG (2013) The family of mapreduce and large-scale data processing systems. *ACM Comput Surv* 46(1):1. <https://doi.org/10.1145/2522968.2522979>
- Sena B, Allian AP, Nakagawa EY (2017) Characterizing big data software architectures: a systematic mapping study. In: Proceeding of the 11th Brazilian symposium on software components, architectures, and reuse (SB-CARS '17). <https://doi.org/10.1145/3132498.3132510>
- Shekhar S, Gunturi V, Evans MR, Yang KS. 2012. Spatial big-data challenges intersecting mobility and cloud computing. In: Proceedings of the eleventh ACM international workshop on data engineering for wireless and mobile access (MobiDE '12), pp 1–6. <https://doi.org/10.1145/2258056.2258058>
- Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). <https://doi.org/10.1109/MSST.2010.5496972>
- Sriharsha R (2015) Magellan: geospatial analytics on spark. <https://hortonworks.com/blog/magellan-geospatial-analytics-in-spark/>. Accessed June 2017
- Tang M, Yu Y, Malluhi QM, Ouzzani M, Aref WG (2016) LocationSpark: a distributed in-memory data management system for big spatial data. *Proc VLDB Endow* 9(13):1565–1568. <https://doi.org/10.14778/3007263.3007310>
- Whitman RT, Park MB, Ambrose SM, Hoel EG (2014) Spatial indexing and analytics on Hadoop. In: Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems (SIGSPATIAL '14), pp 73–82. <https://doi.org/10.1145/2666310.2666387>
- Whitman RT, Park MB, Marsh BG, Hoel EG (2017) Spatio-temporal join on Apache spark. In: Hoel E, Newsam S, Ravada S, Tamassia R, Trajcevski G (eds) Proceedings of the 25th ACM SIGSPATIAL international conference on advances in geographic information systems (SIGSPATIAL'17). <https://doi.org/10.1145/3139958.3139963>
- Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: efficient in-memory spatial analytics. In: Proceedings of the 2016 international conference on management of data (SIGMOD '16), pp 1071–1085. <https://doi.org/10.1145/2882903.2915237>
- You S, Zhang J, Gruenwald L (2015) Large-scale spatial join query processing in Cloud. In: 2015 31st IEEE international conference on data engineering workshops, Seoul, 13–17 April 2015, pp 34–41
- Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems, Seattle, WA
- Yuan Y, Salmi MF, Huai Y, Wang K, Lee R, Zhang X (2016) Spark-GPU: an accelerated in-memory data processing engine on clusters. In: Proceedings of the 2016 IEEE international conference on big data (Big Data 2016), Washington, DC, pp 273–283
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing (HotCloud'10), Boston, MA
- Zhang S, Han J, Liu Z, Wang K, Xu Z (2009) SJMR: parallelizing spatial join with mapreduce on clusters. In: IEEE international conference on Cluster computing (CLUSTER'09), New Orleans, Louisiana, pp 1–8

Artifact-Centric Process Mining

Dirk Fahland
Eindhoven University of Technology,
Eindhoven, The Netherlands

Synonyms

Multi-instance process mining; Object-centric process mining

Definitions

Artifact-centric process mining is an extension of classical process mining (van der Aalst 2016) that allows to analyze event data with more than one case identifier in its entirety. It allows to analyze the dynamic behavior of (business) processes that create, read, update, and delete multiple data objects that are related to each other in relationships with one-to-one, one-to-many, and many-to-many cardinalities. Such event data is typically stored in relational databases of, for example, Enterprise Resource Planning (ERP) systems (Lu et al. 2015). Artifact-centric process mining comprises artifact-centric process discovery, conformance checking, and enhancement. The outcomes of artifact-centric process mining can be used for documenting the actual data flow in an organization and for analyzing deviations in the data flow for performance and conformance analysis.

The input to *artifact-centric process discovery* is either an event log where events carry information about the data objects and their changes, or a relational database also containing records about data creation, change, and deletion events. The output of artifact-centric process discovery is a data model of the objects (each defining its own case identifier) and relations between objects and an artifact-centric process model, as illustrated in Fig. 1. An artifact-centric process model describes the dynamics of each data object on its own in an *object life-cycle model*, and the

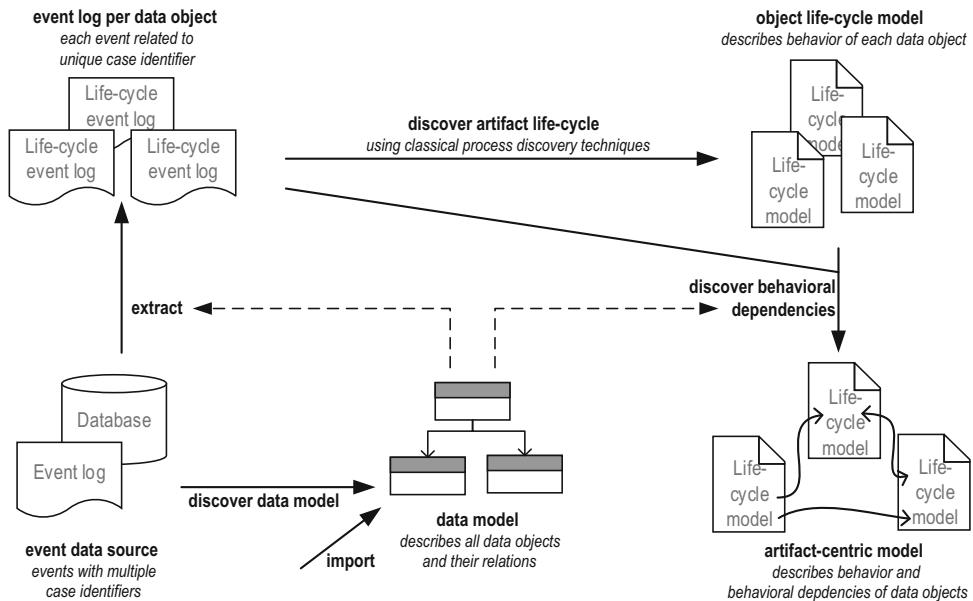
behavioral dependencies between the different data objects. To this end, artifact-centric process discovery integrates the control-flow analysis of event data of classical process mining with an analysis of the data structures and data records related to the events.

During artifact-centric process discovery, each event is associated with one data object in the data source. From the behavioral relations between all events associated with one data object, a *life-cycle model* of the data object is learned using automated process discovery techniques. Each life-cycle model describes the possible changes to the object and their ordering as they have been observed in reality. From behavioral relationships between events in different related data objects, information about *behavioral dependencies between changes in different data objects* is discovered preserving the one-to-one, one-to-many, and many-to-many cardinalities.

Several modeling languages have been proposed to describe a complete artifact-centric model of all object life cycles and their behavioral interdependencies. Existing behavioral modeling languages can be extended to express interdependencies of one-to-many and many-to-many cardinalities including Petri nets (van der Aalst et al. 2001) and UML (Estañol et al. 2012). Specifically designed languages including the Guard-Stage-Milestone (GSM) model (Hull et al. 2011) or data-centric dynamic systems (Hariri et al. 2013) employ both data and behavioral constructs as primary modeling concepts. The Case Management Model and Notation (CMMN) standard v1.1 incorporates several modeling concepts of GSM (OMG 2016).

Artifact-centric conformance checking compares event data to an artifact-centric model with the aim to identify where recorded events *deviate* from the behavior described in the artifact-centric model. Deviations may exist between observed and specified data models, between observed events and the life-cycle model of an artifact, and between observed events and the interactions of two or more artifacts.

Artifact-centric model enhancement uses event data to enrich an artifact-centric model, for example, with information about the frequency of



Artifact-Centric Process Mining, Fig. 1 Overview on artifact-centric process discovery

paths through a life-cycle model or interactions, or to identify infrequent behavior as outliers.

Overview

Historically, artifact-centric process mining addressed the unsolved problem of process mining on event data with multiple case identifiers and one-to-many and many-to-many relationships by adopting the concept of a (business) *artifact* as an alternative approach to describing business processes.

Event Data with Multiple Case Identifiers

Processes in organizations are typically supported by information systems to structure the information handled in these processes in well-defined data objects which are often stored in relational databases. During process execution, various data objects are created, read, updated, and deleted, and various data objects are related to each other in one-to-one, one-to-many, and many-to-many relations. Figure 2 illustrates in a simplified form the data structures typically found in ERP systems. *Sales*, *Delivery*, and

Billing documents are recorded in tables; relation *F1* links *Sales* to *Delivery* documents in a one-to-many relation: *S1* relates to *D1* and *D2*; correspondingly *F2* links *Billing* to *Delivery* documents in a one-to-many relation. Events on process steps and data access are recorded in time stamp attributes such as *Date created* or in a separate *Document Changes* table linked to all other tables.

Convergence and Divergence

Process mining requires to associate events to a *case identifier* in order to analyze behavioral relations between events in the same case (van der Aalst 2016). The data in Fig. 2 provides three case identifiers: *SD id*, *DD id*, and *BD id*. Classical process mining forces to associate all events to a single case identifier. However, this is equivalent to flattening and de-normalizing the relational structure along its one-to-many relationships.

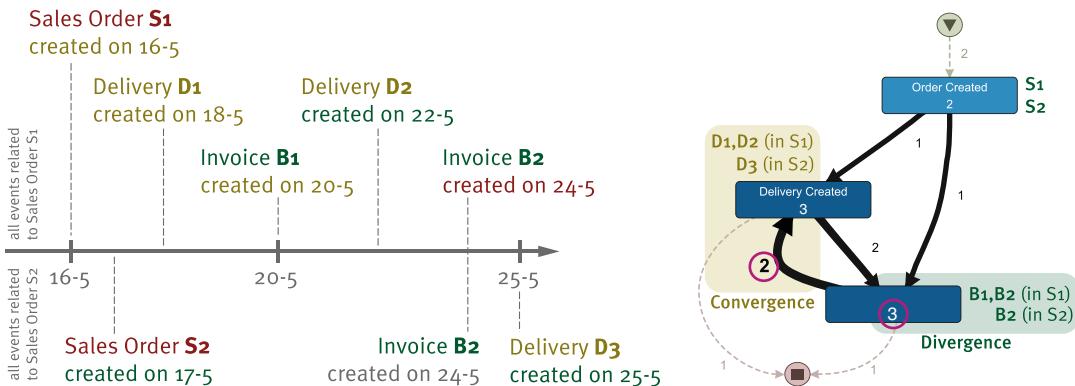
For example, associating all *Create* events of Fig. 2 to *SD id* flattens the tables into the *event log* of Fig. 3(left) having two cases for *S1* and *S2*. Due to flattening, event *Create* of *B2* has been duplicated as it was extracted once for *S1* and once for *S2*, also called *divergence*. Further,

Sales Documents				Billing Documents		
SD id	Date created	Value	Last Change	BD id	Date created	Clearing Date
S1	16-5-2020	100	10-6-2020	B1	20-5-2020	31-5-2020
S2	17-5-2020	200	5-6-2020	B2	24-5-2020	5-6-2020

Delivery Documents				
DD id	Date created	Reference SD id	Reference BD id	Picking Date
D1	18-5-2020	S1	B1	31-5-2020
D2	22-5-2020	S1	B2	5-6-2020
D3	25-5-2020	S2	B2	5-6-2020

Document Changes						
Change id	Date	Ref. id	Table	Change type	Old Value	New Value
1	17-5-2020	S1	SD	Price updated	100	80
2	19-5-2020	S1	SD	Delivery block released	X	-
3	19-5-2020	S1	SD	Billing block released	X	-
4	10-6-2020	B1	BD	Invoice date updated	20-6-2020	21-6-2020

Artifact-Centric Process Mining, Fig. 2 Event data stored in a relational database



Artifact-Centric Process Mining, Fig. 3 An event log (left) serializing the “create” events of the database of Fig. 2 based on the case identifier “SD id.” The resulting directly-follows relation (right) suffers convergence and divergence

Create for *B1* is followed by Create for *D2* although *B1* and *D2* are unrelated in Fig. 2, also called *convergence* (Lu et al. 2015). The behavioral relations which underly automated process discovery become erroneous through convergence and divergence. For example, the directly-follows relation of the log (Fig. 3 right) states erroneously that three *Invoice* documents have been created – whereas the original data source contains only two – and that in two cases *Invoice* creation was followed by *Delivery* creation (between related data objects), whereas in the original data source this only happened once for *B2* and *D3*.

Convergence and divergence may cause up to 50% of erroneous behavioral relations (Lu et al. 2015) in event logs. Convergence and divergence can be avoided partially by scoping extraction of event data into event logs with a single case identifier in a manual process (Jans 2017).

Artifact-Centric Process Models

Artifact-centric process mining adopts modeling concept of a (business) artifact to analyze event data with multiple case identifiers in their entirety (Lu et al. 2015; Nooijen et al. 2012; van Eck et al. 2017). The notion of a (business) *artifact*

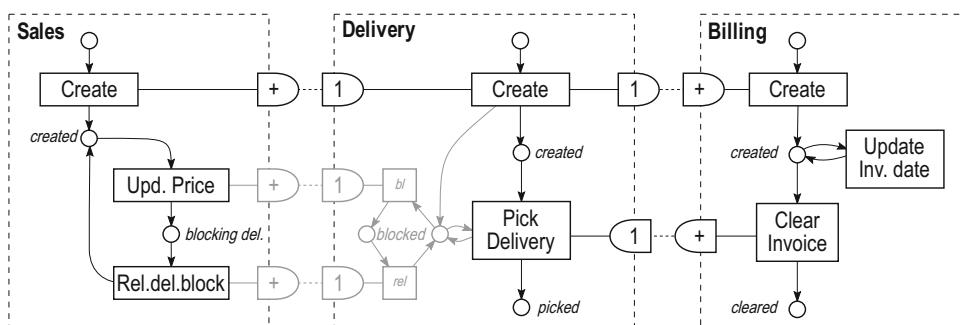
was proposed by Nigam and Caswell (2003) as an alternative approach to describing business processes. This approach assumes that any process materializes itself in the (data) objects that are involved in the process, for instance, sales documents and delivery documents; these objects have properties such as the values of the fields of a paper form, the processing state of an order, or the location of a package. Typically, a *data model* describes the (1) classes of objects that are relevant in the process, (2) the relevant properties of these objects in terms of class attributes, and (3) the relations between the classes. A process execution instantiates new objects and changes their properties according to the process logic. Thereby, the relations between classes describe how many objects of one class are related to how many objects of another class.

An *artifact-centric process model* enriches the classes of the data model themselves with process logic restricting how objects may evolve during execution. More precisely, one *artifact* (1) encapsulates several classes of the data model (e.g., *Sales Documents* and *Sales Document Lines*), (2) provides *actions* that can update the classes attributes and move the artifact to a particular state, and (3) defines a *life cycle*. The artifact life cycle describes when an instance of the artifact (i.e., a concrete object) is created, in which state of the instance which actions may occur to advance the instance to another state (e.g., from *created* to *cleared*), and which goal state the instance has to reach to complete a case. A complete artifact-centric process model provides a life-cycle model for each artifact in the process and

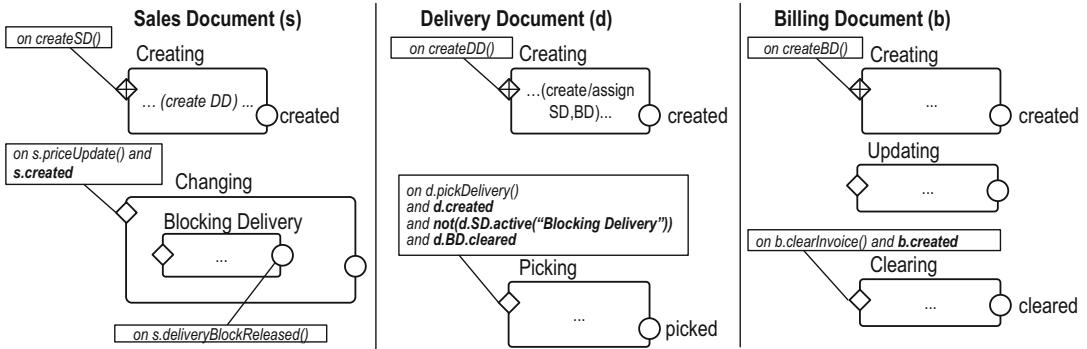
describes which *behavioral dependencies* exist between actions and states of different artifacts (e.g., *pick delivery* may occur for a *Delivery* object only if all its *Billing* objects are in state *cleared*). Where business process models created in languages such as BPMN, EPC, or Petri nets describe a process in terms of activities and their ordering in a single case, an artifact-centric model describes process behavior in terms of creation and evolution of instances of multiple related data objects. In an artifact-centric process model, the unit of modularization is the *artifact*, consisting of data and behavior, whereas in an activity-centric process modeling notation, the unit of modularization is the *activity*, which can be an elementary task or a sub-process. A separate entry in this encyclopedia discusses the problem of automated discovery of activity-centric process models with sub-processes.

Figure 4 shows an artifact-centric process model in the notation of *Proclcts* (van der Aalst et al. 2001) for the database of Fig. 2. The life cycle of each document (*Sales*, *Delivery*, *Billing*) is described as a Petri net. Behavioral dependencies between actions in different objects are described through *interface ports* and *asynchronous channels* that also express cardinalities in the interaction. For example, the port annotation “+” specifies that *Clear Invoice* in *Billing* enables *Pick Delivery* in *multiple related Delivery* objects. Port annotation “1” specifies that *Pick Delivery* can occur after *Clear Invoice* occurred in the *one Billing* object related to the *Delivery*.

The gray part of Fig. 4 shows a more involved behavioral dependency. Whenever *Update Price*



Artifact-Centric Process Mining, Fig. 4 Example of an artifact-centric process model in the proclct notation



Artifact-Centric Process Mining, Fig. 5 Example of an artifact-centric process model in the Guard-Stage-Milestone notation

occurs in a *Sales* document, all related *Delivery* documents get *blocked*. Only after *Release delivery block* occurred in *Sales*, the *Delivery* document may be updated again, and *Pick Delivery* may occur. For the sake of simplicity, the model does not show further behavioral dependencies such as “*Update price* also blocks related *Billing* documents.”

Figure 5 shows the same model in the *Guard-Stage-Milestone* notation (Hull et al. 2011) (omitting some details). Each round rectangle denotes a *stage* that can be entered when its *guard* condition (diamond) holds and is left when its *milestone* condition (circle) holds. The guard and milestone conditions specify declarative constraints over data attributes, stages, and milestones of *all* artifacts in the model. For example, *Picking* can start when the *pickDelivery* event is triggered in the process, the delivery document has reached its *created* milestone, the billing document related to the delivery document has reached its *cleared* milestone (*d.BD.cleared*), and the stage *Blocking Delivery* is not active in the related sales document.

Artifact-Centric Process Mining

The behavior recorded in the database of Fig. 2 does not *conform* to the models in Figs. 4 and 5:

1. *Structural conformance* states how well the data model describes the data records observed in reality. The proclet model of Fig. 4 structurally conforms to the data in Fig. 2

regarding objects and relations but not regarding actions: the recorded event data shows two additional event types for the life cycle of the *Sales* document – *Release billing block* and *Last Change*.

2. *Life-cycle conformance* states how well the life-cycle model of each artifact describes the order of events observed in reality. This corresponds to conformance in classical process mining. For example, in Fig. 2, *Update invoice date* occurs in *Billing* after *Clear Invoice* which does not conform to the life-cycle model in Fig. 4.
3. *Interaction conformance* states how well the entire artifact centric model describes the behavioral dependencies between artifacts. In Fig. 2, instance *D3* of *Delivery* is *created* after its related instance *B2* of *Billing*. This does not conform to the channels and ports specified in Fig. 4.

The objective of artifact-centric process mining is to relate recorded behavior to modeled behavior, through (1) *discovering* an artifact-centric process model that conforms to the recorded behavior, (2) *checking* how well recorded behavior and an artifact-centric model *conform* to each other and detecting *deviations*, and (3) extending a given artifact-centric model with further information based on recorded event data.

Artifact-centric process discovery is a technique to automatically or semiautomatically learn artifact-centric process models from event data.

The problem is typically solved by a decomposition into the following four steps:

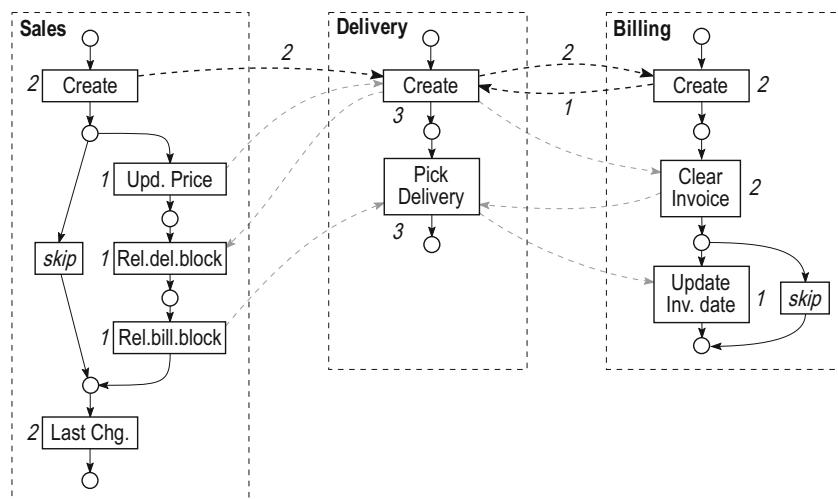
1. Discovering the data model of entities or tables, their attributes, and relations from the data records in the source data. This step corresponds to *data schema recovery*. It can be omitted if the data schema is available and correct; however, in practice foreign key relations may not be documented at the data level and need to be discovered.
2. Discovering artifact types and relations from the data model and the event data. This step corresponds to transforming the data schema discovered in step 1 into a domain model often involving undoing horizontal and vertical (anti-) partitioning in the technical data schema and grouping entities into domain-level data objects. User input or detailed information about the domain model are usually required.
3. Discovering artifact life-cycle models for each artifact type discovered in step 2. This step corresponds to automated process discovery for event data with a single case identifier and can be done fully automatically up to parameters of the discovery algorithm.
4. Discovering behavioral dependencies between the artifact life cycles discovered in step 3

based on the relations between artifact types discovered in step 2. This step is specific to artifact-centric process mining; several alternative, automated techniques have been proposed. User input may be required to select domain-relevant behavioral dependencies among the discovered ones.

In case the original data source is a relational database, steps 3 and 4 require to *automatically extract event logs* from the data source for discovering life-cycle models and behavioral dependencies. As in classical process discovery, it depends on the use case to which degree the discovered data model, life-cycle model, and behavioral dependencies shall conform to the original data.

Artifact-centric conformance checking and *Artifact-centric model enhancement* follow the same problem decomposition into data schema, artifact types, life cycles, and interactions as artifact-centric discovery. Depending on which models are available, the techniques may also be combined by first discovering data schema and artifact types, then extracting event logs, and then checking life-cycle and behavioral conformance for an existing model or enhancing an existing artifact model with performance information.

Figure 6 shows a possible result of artifact-centric process discovery on the event data in



Artifact-Centric Process Mining, Fig. 6 Possible result of artifact-centric process discovery from the event data in Fig. 2

Fig. 2 using the technique of Lu et al. (2015) where the model has been enhanced with information about the *frequencies* of occurrences of events and behavioral dependencies.

Key Research Findings

Artifact-type discovery. Nooijen et al. (2012) provide a technique for automatically discovering artifact types from a relational database, leveraging schema summarization techniques to cluster tables into artifact types based on information entropy in a table and the strength of foreign key relations. The semiautomatic approach of Lu et al. (2015) can then be used to refine artifact types and undo horizontal and vertical (anti-) partitioning and to discover relations between artifacts. Popova et al. (2015) show how to discover artifact types from a rich event stream by grouping events based on common identifiers into entities and then deriving structural relations between them.

Event log extraction. In addition to discovering artifact types, Nooijen et al. (2012) also automatically create a mapping from the relational database to the artifact-type specification. The technique of Verbeek et al. (2010) can use this mapping to generate queries for event log extraction for life-cycle discovery automatically. Jans (2017) provides guidelines for extracting specific event logs from databases through user-defined queries. The event log may also be extracted from database redo logs using the technique of de Murillas et al. (2015) and from databases through a meta-model-based approach as proposed by de Murillas et al. (2016).

Life-cycle discovery. Given the event log of an artifact, artifact life-cycle discovery is a classical automated process discovery problem for which various process discovery algorithms are available, most returning models based on or similar to Petri nets. Weerdt et al. (2012) compared various discovery algorithms using real-life event logs. Lu et al. (2015) advocates the use of the

Heuristics Miner of Weijters and Ribeiro (2011) and vanden Broucke and Weerdt (2017) with the aim of visual analytics. Popova et al. (2015) advocate to discover models with precise semantics and free of behavioral anomalies that (largely) fit the event log (Leemans et al. 2013; Buijs et al. 2012) allowing for translating the result to the Guard-Stage-Milestone notation.

Behavioral dependencies. Lu et al. (2015) discover behavioral dependencies between two artifacts by extracting an *interaction event log* that combines the events of any two related artifact instances into one trace. Applying process discovery on this interaction event log then allows to extract “flow edges” between activities of the different artifacts, also across one-to-many relations, leading to a model as shown in Fig. 6. This approach has been validated to return only those dependencies actually recorded in the event data but suffers when interactions can occur in many different variants, leading to many different “flow edges.”

van Eck et al. (2017) generalize the interaction event log further and create an integrated event log of all artifact types to be considered (two or more) where for each combination of related artifact instances, all events are merged into a single trace. From this log, a composite state machine model is discovered which describes the synchronization of all artifact types. By projecting the composite state machine onto the steps of each artifact type, the life-cycle model for each artifact is obtained, and the interaction between multiple artifacts can be explored interactively in a graphical user interface through their relation in the composite state machine. This approach assumes one-to-one relations between artifacts.

Popova and Dumas (2013) discover behavioral dependencies in the form of data conditions over data attributes and states of other artifacts, similar to the notation in Fig. 5 but is limited to one-to-one relations between artifacts.

Conformance checking. Artifact life-cycle conformance can be checked through extracting artifact life-cycle event logs and then applying classical conformance checking techniques (Fahland

et al. 2011a). The technique in Fahland et al. (2011b) checks interaction conformance in an artifact life-cycle model if detailed information about which artifact instances interact is recorded in the event data.

Models for artifacts. Artifact-centric process mining techniques originated and are to a large extent determined by the modeling concepts available to describe process behavior and data flow with multiple case identifiers. Several proposals have been made in this area. The Procler notation (van der Aalst et al. 2001) extended Petri nets with ports that specify one-to-many and many-to-many cardinality constraints on messages exchanged over channels in an asynchronous fashion. Fahland et al. (2011c) discuss a normal form for procler-based models akin to the second normal form in relational schemas. The Guard-Stage-Milestone (GSM) notation (Hull et al. 2011) allows to specify artifacts and interactions using event-condition-actions rules over the data models of the different artifacts. Several modeling concepts of GSM were adopted by the CMMN 1.1 standard of OMG (2016). Hariri et al. (2013) propose data-centric dynamic systems (DCDS) to specify artifact-centric behavior in terms of updates of database records using logical constraints. Existing industrial standards can also be extended to describe artifacts as shown by Lohmann and Nyolt (2011) for BPMN and by Estañol et al. (2012) for UML. Freedom of behavioral anomalies can be verified for UML-based models (Calvanese et al. 2014) and for DCDS (Montali and Calvanese 2016). Meyer and Weske (2013) show how to translate between artifact-centric and activity-centric process models, and Lohmann (2011) shows how to derive an activity-centric process model describing the interactions between different artifacts based on behavioral constraints.

Examples of Application

Artifact-centric process mining is designed for analyzing event data where events can be related

to more than one case identifier or object and where more than one case identifier has to be considered in the analysis.

The primary use case is in analyzing processes in information systems storing multiple, related data objects, such as Enterprise Resource Planning (ERP) systems. These systems store documents about business transactions that are related to each other in one-to-many and many-to-many relations. Lu et al. (2015) correctly distinguish normal and outlier flows between 18 different business objects over 2 months of data of the Order-to-Cash process in an SAP ERP system using artifact-centric process mining. The same technique was also used for identifying outlier behavior in processes of a project management system together with end users. van Eck et al. (2017) analyzed the personal loan and overdraft process of a Dutch financial institution. Artifact-centric process mining has also been applied successfully on software project management systems such as Jira and customer relationship management systems such as Salesforce (Calvo 2017).

Artifact-centric process mining can also be applied on event data outside information systems. One general application area is analyzing the behavior of physical objects as sensed by multiple related sensors. For instance, van Eck et al. (2016) analyzed the usage of physical objects equipped with multiple sensors. Another general application area is analyzing the behavior of software components from software execution event logs. For instance, Liu et al. (2016) follow the artifact-centric paradigm to structure events of software execution logs into different components and discover behavioral models for each software component individually.

Future Directions for Research

At the current stage, artifact-centric process mining is still under development allowing for several directions for future research.

Automatically discovering artifact types from data sources is currently limited to summarizing the structures in the available data. Mapping these

structures to domain concepts still requires user input. Also the automated extraction of event logs from the data source relies on the mapping from the data source to the artifact-type definition. How to aid the user in discovering and mapping the data to domain-relevant structures and reducing the time and effort to extract event logs, possibly through the use of ontologies, is an open problem. Also little research has been done for improving the queries generated for automated event log extraction to handle large amount of event data.

For discovering behavioral dependencies between artifacts, only few and limited techniques are available. The flow-based discovery of Lu et al. (2015) that can handle one-to-many relations is limited to interactions between two artifacts and suffers in the presence of many different behavioral variants of the artifacts or the interactions. The alternative approaches (Popova and Dumas 2013; van Eck et al. 2017) are currently limited to one-to-one relations between artifacts. Solving the discovery of behavioral dependencies between artifacts thereby faces two fundamental challenges:

1. Although many different modeling languages and concepts for describing artifact-centric processes have been proposed, the proposed concepts do not adequately capture these complex dynamics in an easy-to-understand form (Reijers et al. 2015). Further research is needed to identify appropriate modeling concepts for artifact interactions.
2. Systems with multiple case identifiers are in their nature complex systems, where complex behaviors and multiple variants in the different artifacts multiply when considering artifact interactions. Further research is needed on how to handle this complexity, for example, through generating specific, interactive views as proposed by van Eck et al. (2017).

Although several, comprehensive conformance criteria in artifact-centric process mining have been identified, only behavioral conformance of artifact life cycles can currently be measured. Further research for measuring

structural conformance and interaction conformance is required, not only for detecting deviations but also to objectively evaluate the quality of artifact-centric process discovery algorithms.

Cross-References

- [Automated Process Discovery](#)
- [Business Process Analytics](#)
- [Conformance Checking](#)
- [Hierarchical Process Discovery](#)
- [Multidimensional Process Analytics](#)
- [Schema Mapping](#)

References

- Buijs JCAM, van Dongen BF, van der Aalst WMP (2012) A genetic algorithm for discovering process trees. In: IEEE congress on evolutionary computation
- Calvanese D, Montali M, Estañol M, Teniente E (2014) Verifiable UML artifact-centric business process models. In: CIKM
- Calvo HAS (2017) Artifact-centric log extraction for cloud systems. Master's thesis, Eindhoven University of Technology
- de Murillas EGL, van der Aalst WMP, Reijers HA (2015) Process mining on databases: unearthing historical data from redo logs. In: BPM
- de Murillas EGL, Reijers HA, van der Aalst WMP (2016) Connecting databases with process mining: a meta model and toolset. In: BMMDS/EMMSAD
- Estañol M, Queralt A, Sancho MR, Teniente E (2012) Artifact-centric business process models in UML. In: Business process management workshops
- Fahland D, de Leoni M, van Dongen BF, van der Aalst WMP (2011a) Behavioral conformance of artifact-centric process models. In: BIS
- Fahland D, de Leoni M, van Dongen BF, van der Aalst WMP (2011b) Conformance checking of interacting processes with overlapping instances. In: BPM
- Fahland D, de Leoni M, van Dongen BF, van der Aalst WMP (2011c) Many-to-many: some observations on interactions in artifact choreographies. In: ZEUS
- Hariri BB, Calvanese D, Giacomo GD, Deutsch A, Montali M (2013) Verification of relational data-centric dynamic systems with external services. In: PODS
- Hull R, Damaggio E, Masellis RD, Fournier F, Gupta M, Heath FT, Hobson S, Linehan MH, Maradugu S, Nigam A, Sukaviriy A, Vaculín R (2011) Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS

- Jans M (2017) From relational database to event log: decisions with quality impact. In: First international workshop on quality data for process analytics
- Leemans SJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs – a constructive approach. In: Petri nets
- Liu CS, van Dongen BF, Assy N, van der Aalst WMP (2016) Component behavior discovery from software execution data. In: 2016 IEEE symposium series on computational intelligence (SSCI), pp 1–8
- Lohmann N (2011) Compliance by design for artifact-centric business processes. *Inf Syst* 38(4): 606–618
- Lohmann N, Nyolt M (2011) Artifact-centric modeling using BPMN. In: ICSOC workshops
- Lu X, Nagelkerke M, van de Wiel D, Fahland D (2015) Discovering interacting artifacts from ERP systems. *IEEE Trans Serv Comput* 8:861–873
- Meyer A, Weske M (2013) Activity-centric and artifact-centric process model roundtrip. In: Business process management workshops
- Montali M, Calvanese D (2016) Soundness of data-aware, case-centric processes. *Int J Softw Tools Technol Trans* 18:535–558
- Nigam A, Caswell NS (2003) Business artifacts: an approach to operational specification. *IBM Syst J* 42: 428–445
- Nooijen EHJ, van Dongen BF, Fahland D (2012) Automatic discovery of data-centric and artifact-centric processes. In: Business process management workshops. Lecture notes in business information processing, vol 132. Springer, pp 316–327. https://doi.org/10.1007/978-3-642-36285-9_36
- OMG (2016) Case management model and notation, version 1.1. <http://www.omg.org/spec/CMMN/1.1>
- Popova V, Dumas M (2013) Discovering unbounded synchronization conditions in artifact-centric process models. In: Business process management workshops
- Popova V, Fahland D, Dumas M (2015) Artifact lifecycle discovery. *Int J Coop Inf Syst* 24:44
- Reijers HA, Vanderfeesten ITP, Plomp MGA, Gorp PV, Fahland D, van der Crommert WLM, Garcia HDD (2015) Evaluating data-centric process approaches: does the human factor factor in? *Softw Syst Model* 16:649–662
- vanden Broucke SKLM, Weerdt JD (2017) Fodina: a robust and flexible heuristic process discovery technique. *Decis Support Syst* 100:109–118
- van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer. <https://doi.org/10.1007/978-3-662-49851-4>
- van der Aalst WMP, Barthelmeß P, Ellis CA, Wainer J (2001) Proclcts: a framework for lightweight interacting workflow processes. *Int J Coop Inf Syst* 10: 443–481
- van Eck ML, Sidorova N, van der Aalst WMP (2016) Composite state machine miner: discovering and exploring multi-perspective processes. In: BPM
- van Eck ML, Sidorova N, van der Aalst WMP (2017) Guided interaction exploration in artifact-centric pro-
- cess models. In: 2017 IEEE 19th conference on business informatics (CBI), vol 1, pp 109–118
- Verbeek HMW, Buijs JCAM, van Dongen BF, van der Aalst WMP (2010) Xes, xesame, and prom 6. In: CAiSE forum
- Weerdt JD, Backer MD, Vanthienen J, Baesens B (2012) A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf Syst* 37(7):654–676. <https://doi.org/10.1016/j.is.2012.02.004>
- Weijters AJMM, Ribeiro JTS (2011) Flexible heuristics miner (FHM). In: 2011 IEEE symposium on computational intelligence and data mining (CIDM), pp 310–317

Assessment

► Auditing

Attestation

► Auditing

Attribute-Based Access Control (ABAC)

► Security and Privacy in Big Data Environment

Auditing

Francois Raab
InfoSizing, Manitou Springs, CO, USA

Synonyms

Assessment; Attestation; Certification; Review; Validation

Definitions

An examination of the implementation, execution, and results of a test or benchmark, generally

performed by an independent third-party, and resulting in a report of findings

Historical Background

The use of a third-party audit to attest to the veracity of a claim has historically been commonplace in the financial sector. The goal of such audit activities is to bolster the credibility of an organization's claims regarding its financial standing. Similar auditing activities are found in other fields where there is value in validating the level at which a set of requirements have been followed.

As formal definitions of computer systems performance benchmarks started to emerge, so did the call for independent certification of published results. The Transaction Processing Performance Council (TPC – www.tpc.org) was the first industry standard benchmark consortium to formalize the requirement for independent auditing of benchmark results.

Foundations

The purpose of auditing in the context of a performance test or benchmark is to validate that the test results were produced in compliance with the set of requirements defining the test. These requirements are used to define multiple aspects of the test, including what is being tested, how it is being tested, how the test results are measured, and how accurately are they documented. Auditing a test or benchmark result consists in validating some or all of these requirements.

Benchmark requirements can be viewed as belonging to one of the following categories: implementation rules, execution rules, results collection, pricing rules, and documentation. The motivation behind auditing a set of requirements, and the process involved in such validation, is largely based on which of these categories the requirements belong to.

Auditing the Implementation

Some benchmarks are provided in the form of a complete software kit that can simply be installed

and executed to measure the underlying system. Other benchmarks are provided in the form of a set of functional requirements to be implemented using any fitting technology. In this later case, the level at which the implementation meets the stated requirements can directly affect the results of the test. Consider a benchmark requiring the execution of two tasks acting on the same data set. An implementation that correctly implements the tasks but fails to have them act on the same data set would avoid potential data access conflicts.

The process of auditing a benchmark implementation includes all aspects of that implementation. This may include reviewing custom code, examining data generation tools and their output, executing functional testing to verify the proper behavior of required features, and validating the integration of the various benchmark components.

Auditing the Execution

Most benchmarks involve the execution of multiple steps, for a prescribed duration and in some specified sequence. The level at which these execution rules are followed can greatly impact the outcome of the test. Consider a benchmark requiring that two tasks be executed concurrently for a specified duration. An execution that runs the tasks for the specified duration but schedules them in a serial manner would avoid potential contention for system resources.

The process of auditing a benchmark execution involves verifying that controls are in place to drive the execution based on the stated rules and that sufficient traces of the execution steps are captured. This may be accomplished by reviewing execution scripts, witnessing the execution in real time, and examining logs that were produced during the execution.

Auditing the Results

The end goal of implementing and executing a benchmark is to produce a result for use in performance engineering, marketing, or other venue. While a benchmark may be implemented correctly and executed according to all stated rules, it may produce a flawed outcome if the results

are not collected properly. Consider a benchmark that involves a workload that gradually ramps up until the system under test reaches saturation, with the goal of measuring the system's behavior at that saturation point. A test that measures the system's behavior too early during ramp-up would avoid the potential disruptions caused by saturation.

The process of auditing the collection of results during a benchmark execution involves verifying that all necessary conditions are met for the measurement to be taken. In cases where metrics are computed from combining multiple measurements, it also involves verifying that all the components of the metric are properly sourced and carry sufficient precision.

Auditing the Pricing

When benchmarks are executed for the purpose of competitive analysis, the cost of the tested configuration may also be part of the benchmark's metrics. Including a cost component in the metrics provides a measure of value, in addition to the measure of performance. This value metric may help differentiate between systems under test with comparable performance metrics. But the price of a SUT can be greatly influenced by the rules regulating the pricing methodology. For instance, the price of a leasing contract may not be comparable to that of an outright purchase, discount levels may vary widely across markets, and maintenance costs vary based on service level agreements.

The process of auditing the pricing of a SUT involved in a benchmark execution consists in verifying that all of the pricing rules defined by the benchmark have been adhered to. It may also involve a verification of the stated prices by attempting to obtain an independent pricing quotation from the vendors or distributors of the components in the SUT.

Auditing the Documentation

Benchmark results are best understood within their proper context. The purpose of documenting a benchmark result is to provide sufficient context to allow a full understanding of the significance of the result. Without proper context, a

benchmark result can be misunderstood and lead to incorrect conclusions. Consider a benchmark with a body of existing results that have been produced on single-socket systems. A new, record-breaking result may be received more favorably if the documentation omits to disclose that it was produced on a dual-socket system, leaving the reader to assume that a single socket was used.

The process of auditing the documentation of a benchmark result consists in verifying that all relevant information has been provided with accuracy and candor. A criterion to determine if the documentation of a test result is sufficient is when the information provided allows another party to independently reproduce the result.

Key Applications

First Formal Use

According to Serlin (1993), one of the first formal audit of a benchmark result took place "in March, 1987, when Tandem hired Codd & Date Consulting to certify the 208 tps result obtained under a version of DebitCredit." Tom Sawyer, the auditor on this tandem test, later collaborated with Ormi Serlin to author a proposal for a formal performance test framework that became the impetus for the creation of the TPC.

Around the time of the release of its third standard benchmark specification, TPC Benchmark™ C (Raab 1993), in August 1992, the TPC added to its policies (TPC. TPC Policies) the call for a mandatory audit of all its benchmark results. This addition to its policies also established a process by which the TPC certifies individuals who are qualified to conduct such audits.

Auditing Adoption

The practice of independently auditing benchmark results has been adopted by multiple organizations involved in performance testing. A formal benchmark result auditing requirement was adopted by the Storage Performance Council (SPC – www.storageperformance.org), shortly after its inception in 1998. It is a common practice for system vendors to

publish nonstandard test results in support of performance claims or to document a proof of concept. A number of these publications include an independent certification to bolster the credibility of the results. The practice of independently auditing test results has also been adopted by private and governmental organizations conducting comparative performance testing as part of their technology selection process.

Internal Peer Review

Some standard benchmark organizations, such as the Standard Performance Evaluation Corporation (SPEC – www.spec.org), have opted for a peer review process to validate the benchmark results they publish. This review is conducted internally by members of the organization. The TPC also gives test sponsors the option of a peer review process for results against its benchmarks in a category called “express.” In both of these cases of internal peer review, test results from one vendor are reviewed by competing vendors. The strength of this review process is in the assumption that the reviewers are intimately familiar with the testing requirements and the tested technology by having conducted similar tests themselves. The downside is in the potential for a loss of independence in the validation process.

Independent Third-Party Auditor

To maintain its integrity, auditing is best when conducted by an independent third party that has no conflict of interest with the object of the audit. These conflicts can take many forms in the context of a benchmark audit. Following are a few examples of conflicts of interest that may compromise the independence of validation process: having a vested interest in how the benchmark result may affect the image of a tested product; having a financial stake attached to the outcome of the test; or being directly involved in conducting the test or improving the test results.

Cross-References

- ▶ [Big Data Benchmark](#)
- ▶ [Big Data Architectures](#)
- ▶ [Benchmark Harness](#)
- ▶ [Cloud Big Data Benchmarks](#)
- ▶ [Component Benchmark](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [Energy Benchmarking](#)
- ▶ [Metrics for Big Data Benchmarks](#)
- ▶ [Microbenchmark](#)
- ▶ [TPC](#)

References

- Raab F (1993) Overview of the TPC benchmark C: a complex OLTP benchmark. In: The benchmark handbook for database and transaction processing systems, 2nd edn. Morgan Kaufmann Publishers, San Mateo, pp 131–144
- Serlin O (1993) The history of DebitCredit and the TPC. In: The benchmark handbook for database and transaction processing systems, 2nd edn. Morgan Kaufmann Publishers, San Mateo, pp 21–40
- TPC. TPC policies. http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp

Authenticating Anonymized Data with Domain Signatures

- ▶ [Privacy-Aware Identity Management](#)

Authentication, Authorization, and Accounting (AAA)

- ▶ [Security and Privacy in Big Data Environment](#)

Automated Creation of Infographics

- ▶ [Visualizing Semantic Data](#)

Automated Discovery of Hierarchical Process Models

► Hierarchical Process Discovery

Automated Process Discovery

Sander J. J. Leemans

Queensland University of Technology, Brisbane,
Australia

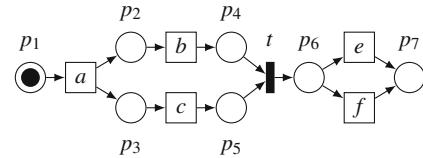
Definitions

An *event log* contains a historical record of the steps taken in a business process. An event log consists of *traces*, one for each case, customer, order, etc. in the process. A trace contains *events*, which represent the steps (*activities*) that were taken for a particular case, customer, order, etc.

An example of an event log derived from an insurance claim handling process is [*{receive claim, check difficulty, decide claim, notify customer}*¹⁰, *{receive claim, check difficulty, check fraud, decide claim, notify customer}*⁵]. This event log consists of 15 traces, corresponding to 15 claims made in the process. In 10 of these traces, the claim was received, its difficulty assessed, the claim was decided and the customer was notified.

A *process model* describes the behaviour that can happen in a process. Typically, it is represented as a Petri net (Reisig 1992) or a BPMN model (OMG 2011).

A Petri net consists of places, which denote the states the system can be in, and transitions, which denote the state changes of the system. For instance, Fig. 1 shows an example of a Petri net. This net starts with a *token* in place p_1 . Firing transition a removes the token from p_1 and puts tokens in p_2 and p_3 . This denotes the execution of the activity a in the process. Then, transitions b and c can fire independently, each consuming the token of p_2 or p_3 and producing a token in p_4 or p_5 . Next, the silent transition t fires and



Automated Process Discovery, Fig. 1 Example of a Petri net

puts a token in p_6 . As t is a silent transition, no corresponding activity is executed in the process. Finally, either e or f can be fired, putting a token in p_7 and ending the process.

A *workflow net* is a Petri net with an initial place (without incoming arcs), a final place (without outgoing arcs) and every place and transition lying on a path between these places. The behaviour of a workflow net is clear: a token is put in the initial place, and every sequence of transitions firings that leads to a token in the final place and nowhere else, is a trace of the behaviour of the net.

A workflow net is *sound* if the net is free of deadlocks, unexecutable transitions and other anomalies (van der Aalst 2016). A workflow net is *relaxed sound* if there is a sequence of transition firings that lead to a token in the final place and nowhere else.

Overview

Automated process discovery aims to extract information from recorded historical information about business processes by means of automatic methods. In this chapter, we discuss challenges and algorithms for process discovery.

Automated Process Discovery

Organisations nowadays store considerable amounts of data: in many business processes such as for booking a flight, lodging an insurance claim or hiring a new employee, every step is supported and recorded by an information system. From these information systems, event logs can be extracted, which contain the steps that were taken for a particular customer, booking,

claim, etc. Process mining aims to derive information and insights from these event logs.

Many process mining techniques depend on the availability of a process model. Process models can be elicited by hand, however this can be a tedious and error-prone task. Instead, if event logs are available, these can be used to discover a process model automatically.

In this chapter, the research field of algorithms that automatically discover process models from event logs is described. First, quality criteria for models are discussed, and how algorithms might have to tradeoff between them. Second, process discovery algorithms are discussed briefly.

Quality Criteria & Tradeoffs

The quality of a discovered model can be assessed using several concepts: whether it possesses clear semantics, whether it is simple, how well it represents the event log and how well it represents the process.

Semantics & Soundness

As a first quality criterion, the behaviour described by the model should be clear. That is, it should be clear which traces the model can produce. If the returned model is a Petri net or a BPMN model, this model should be free of deadlocks, unexecutable transitions and other anomalies (it should be *sound* (van der Aalst 2016)). While unsound nets can be useful for manual analysis, they should be used with care in automated analyses as, for instance, conformance checking techniques might give unreliable answers or simply not work on unsound nets. At a bare minimum, conformance checking techniques such as alignments (Adriansyah 2014) require relaxed sound models.

Simplicity

Second, given two models, all other things equal, the simplest model is usually the best of the two (a principle known as Occam's razor). That is, a model should be as understandable as possible, for instance sma.

Log Quality

Third, one can consider the quality of a discovered model with respect to the event log from which it was discovered, to assess whether the model represents the available information correctly. Typically, besides simplicity, three quality dimensions are considered: fitness, precision and generalisation. Fitness expresses the part of the event log that is captured in the behaviour of the process model. Precision expresses the part of the behaviour of the model that is seen in the event log. Generalisation expresses what part of future behaviour will be likely present in the model.

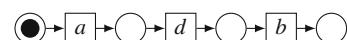
To illustrate these quality measures, consider the following event log L :

$$\begin{aligned} & [\langle a, d, b \rangle^{50}, & \langle a, b, c, d, b \rangle^{20}, \\ & \langle a, b, d, c, b, c, b \rangle^2, & \langle a, b, c, d, b, c, b \rangle^2, \\ & \langle a, b, c, b, d, c, b \rangle, & \langle a, b, c, b, c, b, d \rangle, \\ & \langle a, b, c, b, d, c, b, c, b \rangle, & \langle a, b, c \rangle] \end{aligned}$$

Figure 2 contains a possible process model for L , which supports only a single trace. This model has a poor fitness, as many traces of L are not part of its behaviour. However, it has a high precision, as the single trace it represents was seen in L . Compared to the event log, this model is not very informative.

An extreme model is shown in Fig. 3. This model is a so-called *flower model*, as it allows for all behaviour consisting of a, b, c and d , giving it a low fitness and high precision. Even though this model is simple and certainly generalises, it is completely useless as it does not provide any information besides the presence of $a-d$ in the process.

On the other end of the spectrum is the *trace model*, shown in Fig. 4. This model simply



Automated Process Discovery, Fig. 2 A process model with low fitness, high precision, low generalisation and high simplicity w.r.t. L

lists all traces of L , thereby achieving perfect fitness and precision. However, this model does not generalise the behaviour in the event log, that is, it only shows the traces that were seen in L , and does not provide any extra information.

As a final model, we consider the model shown in Fig. 5. This model has a high fitness, precision, generalisation and simplicity. However, the model still does not score *perfect* as the last trace of L , $\langle a, b, c \rangle$, is not captured by this model, which lowers fitness a bit. Furthermore, precision is not perfect as the trace $\langle a, b, c, b, c, d, b \rangle$ is possible in the model but did not appear in L , which lowers precision.

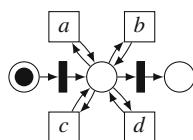
The models shown for L illustrate that process discovery algorithms might have to tradeoff and strike a balance between quality criteria. For some event logs, a model scoring high on all log-quality measures and simplicity might not exist (Buijs et al. 2012b). The necessary balance might depend on the use case at hand. For instance, manual analysis where the “main flow” of a process is sought might require the omission of the last trace of L from the model, yielding a simple and precise model. However, for auditing purposes, one might opt for a perfectly fitting model by including this last trace of L in the behaviour of the model.

Process Quality

A downside of measuring the quality of a model with respect to the event log is that an event log contains only *examples* of behaviour of an (unknown) business process rather than the full behaviour, and that the log might contain traces that do not correspond to the business process (*noisy* traces). Therefore, one can also consider how it compares to the process from which the event log was recorded. In the ideal case, the behaviour of the process is rediscovered by a discovery algorithm. That is, the behaviour (language) of the model is the same as the behaviour of the process.

As the business process is assumed to be unknown, whether an algorithm can find a model that is behaviourally equivalent to the process (*rediscoverability*) is a formal property of the algorithm. Without rediscoverability, an algorithm is *unable* to find a model equivalent to the process, which makes the algorithm rather unsuitable to study this process.

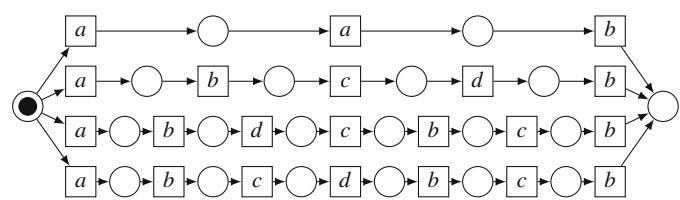
Rediscoverability is typically proven using assumptions on the process and the event log, for instance that it is representable as a model in the formalism of the algorithm (Petri nets, BPMN), and for instance that the event log contains enough information and does not contain too much noise, as well as assumptions on the process.

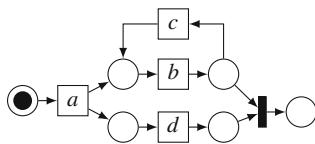


Automated Process Discovery, Fig. 3 A process model (“flower model”) with high fitness, low precision, high generalisation and high simplicity w.r.t. L

Automated Process Discovery, Fig. 4

A process model (“trace model”) with high fitness, high precision, low generalisation and low simplicity w.r.t. L





Automated Process Discovery, Fig. 5 A process model with high fitness, high precision, high generalisation and high simplicity w.r.t. L

For benchmarks and a more exhaustive overview, please refer to Augusto et al. (2017b) (algorithms after 2012) and Weerdt et al. (2012) (algorithms before 2012). Not all algorithms can be benchmarked reliably; the selection here contains all benchmarked algorithms of Augusto et al. (2017b).

Several of these algorithms are available in the ProM framework (van Dongen et al. 2005), which is available for download from <http://www.promtools.org>, or in the Apromore suite (Rosa et al. 2011), which can be accessed via <http://apromore.org>.

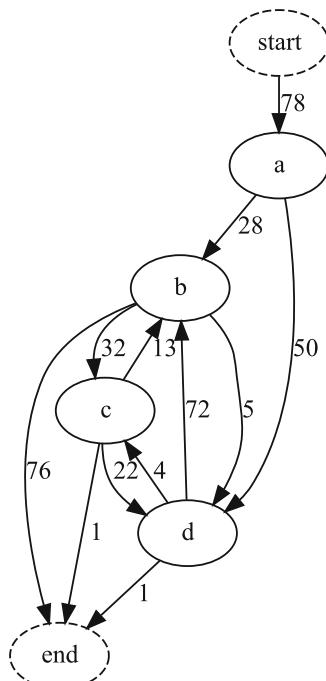
The algorithms are discussed in three stages: firstly, algorithms that do not support concurrency, secondly algorithms that guarantee soundness and thirdly the remaining algorithms.

Directly Follows-Based Techniques

As a first set, techniques based on the directly follows relations are discussed. The section starts with an explanation of directly follows graphs, after which some tools that use this concept are listed and the limitations of such techniques are discussed.

In a directly follows graph, the nodes represent the activities of the event log, and the edges represent that an activity is directly followed by another activity in the event log. Numbers on the edges indicate how often this happened. Additionally, a start and an end node denote the events with which traces in the event log start or end. For instance, Fig. 6 shows the directly follows graph for our event log L .

For more complicated processes, a directly follows graph might get incomprehensibly complicated. Therefore, discovery techniques typically filter the directly follows graph, for instance by removing little-occurring edges. Com-



Automated Process Discovery, Fig. 6 Directly follows graph of event log L

mercial techniques that filter and show directly follows graphs include Fluxicon Disco (Fluxicon 2017), Celonis Process Mining (Celonis 2017) and ProcessGold Enterprise Platform (ProcessGold 2017). Another strategy to reduce complexity, applied by the Fuzzy Miner (Günther and van der Aalst 2007), is to cluster similar activities into groups, thereby providing capabilities to zoom in on details of the process (by clustering less), or to abstract to the main flow of the process by clustering more.

While these graphs are intuitive, it can be challenging to distinguish repetitive and concurrent behaviour, as both manifest as edges forth- and back between activities. For instance, in Fig. 6, it seems that b , c and d can be executed repeatedly, while in the log L this never happened for d . In contrast, it also seems that b , c and d are concurrent, while in L , b and c are always executed repeatedly. Due to this, directly follows graphs tend to have a low precision and high generalisation: in our example, almost any sequence of b , c and d is included.

Nevertheless, directly follows-based techniques are often used to get a first idea of the process behind an event log.

Soundness-Guaranteeing Algorithms

Soundness is a prerequisite for further automated or machine-assisted analysis of business process models. In this section, soundness or relaxed soundness guaranteeing algorithms are discussed.

Evolutionary Tree Miner

To address the issue of soundness, the Evolutionary Tree Miner (ETM) (Buijs et al. 2012a) discovers process trees. A process tree is an abstract hierarchical view of a workflow net and is inherently sound.

ETM first constructs an initial population of models; randomly or from other sources. Second, some models are selected based on fitness, precision, generalisation and simplicity with respect to the event log. Third, the selected models are smart-randomly mutated. This process of selection and mutation is repeated until a satisfactory model is found, or until time runs out.

ETM is flexible as both the selection and the stopping criteria can be adjusted to the use case at hand; one can prioritise (combinations of) quality criteria. However, due to the repeated evaluation of models, on large event logs of complex processes, stopping criteria might force a user to make the decision between speed and quality.

Inductive Miner Family

The Inductive Miner (IM) family of process discovery algorithms, like the Evolutionary Tree Miner, discovers process trees to guarantee that all models that are discovered are sound. The IM algorithms apply a recursive strategy: first, the “most important” behaviour of the event log is identified (such as sequence, exclusive choice, concurrency, loop, etc.). Second, the event log is split in several parts, and these steps are repeated until a base case is encountered (such as a log consisting of a single activity). If no “most important” behaviour can be identified, then the algorithms try to continue the recursion

by generalising the behaviour in the log, in the worst case ultimately ending in a flower model.

Besides a basic IM (Leemans et al. 2013a), algorithms exist that focus on filtering noise (Leemans et al. 2013b), handling incomplete behaviour (when the event log misses crucial information of the process) (Leemans et al. 2014a), handling lifecycle information of events (if the log contains information of e.g. when activities started and ended) (Leemans et al. 2015), discovering challenging constructs such as inclusive choice and silent steps (Leemans 2017), and handling very large logs and complex processes (Leemans et al. 2016), all available in the ProM framework.

Several IM-algorithms guarantee to return a model that perfectly fits the event log, and all algorithms are capable of rediscovering the process, assuming that the process can be described as a process tree (with some other restrictions, such as no duplicated activities) and assuming that the event log contains “enough” information. However, due to the focus on fitness, precision tends to be lower on event logs of highly unstructured processes.

All Inductive Miner algorithms are available as plug-ins of the ProM framework, and some as plug-ins of the Apromore framework. Furthermore, the plug-in Inductive visual Miner (Leemans et al. 2014b) provides an interactive way to apply these algorithms and perform conformance checking.

An algorithm that uses a similar recursive strategy, but lets constructs compete with one another is the Constructs Competition Miner (Redlich et al. 2014), however its implementation has not been published.

Structured Miner

The Structured Miner (STM) (Augusto et al. 2016) applies a different strategy to obtain highly block-structured models and to tradeoff the log quality criteria. Instead of discovering block-structured models directly, SM first discovers BPMN models and, second, structures these models. The models can be obtained from any other discovery technique, for instance Heuristics Miner or Fodina, as these models need not be

sound. These models are translated to BPMN, after which they are made block-structured by shifting BPMN-gateways in or out, thereby duplicating activities.

STM benefits from the flexibility of the used other discovery technique to strike a flexible balance between log-quality criteria and can guarantee to return sound models. However, this guarantee comes at the price of equivalence (the model is changed, not just restructured), simplicity (activities are duplicated) and speed (the restructuring is $O(n^n)$).

STM is available as both a ProM and an Apromore plugin.

(Hybrid) Integer Linear Programming Miner

The Integer Linear Programming Miner (ILP) van der Werf et al. (2009) constructs a Petri net, starting with all activities as transitions and no places, such that every activity can be arbitrarily executed. Second, it adds places using an optimisation technique: a place is only added if it does not remove any trace of the event log from the behaviour of the model. Under this condition, the behaviour is restricted as much as possible.

ILP focusses on fitness and precision: it guarantees to return a model that fits the event log, and the most precise model within its representational bias (Petri nets, no duplicated activities). However, the ILP miner does not guarantee soundness, does not handle noise and tends to return complex models (Leemans 2017).

The first of these two have been addressed in the HybridILPMiner (van Zelst et al. 2017), which performs internal noise filtering. Furthermore, it adjusts the optimisation step to guarantee that the final marking is always reachable, and, in some cases, returns workflow nets, thereby achieving *relaxed soundness*.

Declarative Techniques

Petri nets and BPMN models express what *can* happen when executing the model. In contrast, declarative models, such as Declare models, express what *cannot* happen when executing the model, thereby providing greater flexibility in modelling. Declare miners such as Maggi et al.

(2011), Di Ciccio et al. (2016), and Ferilli et al. (2016) discover the constraints of which Declare models consist using several acceptance criteria, in order to be able to balance precision and fitness. However, using such models in practice tends to be challenging (Augusto et al. 2017b).

Other Algorithms

Unsound models are unsuitable for further automated processing, however might be useful for manual analysis. In the remainder of this section, several algorithms are discussed that do not guarantee soundness.

α -Algorithms

The first process discovery algorithm described was the α -algorithm (van der Aalst et al. 2004). The α algorithm considers the directly follows graph and identifies three types of relations between sets of activities from the graph: sequence, concurrency and mutual exclusivity. From these relations, a Petri net is constructed by searching for certain maximal patterns.

The α algorithm is provably (Badouel 2012) able to rediscover some processes, assuming that the log contains enough information and with restrictions on the process. In later versions, several restrictions have been addressed, such as: (a) no short loops (activities can follow one another directly; addressed in α^+ (de Medeiros et al. 2004)), (b) no long-distance dependencies (choices later in the process depend on choices made earlier; addressed in Wen et al. (2006)), (c) no non-free-choice constructs (transitions that share input places have the same input places; addressed in α^{++} (Wen et al. 2007a)), and (d) no silent transitions (addressed in $\alpha^\#$ (Wen et al. 2007b, 2010) and in $\alpha^\$$ (Guo et al. 2015)). Furthermore, a variant has been proposed, called the Tsinghua- α (Wen et al. 2009), that deals with non-atomic event logs. That is, event logs in which executions of activities take time.

However, these algorithms guarantee neither soundness nor perfect fitness nor perfect precision, the algorithms cannot handle noise and cannot handle incompleteness. Furthermore, the α algorithms might be less fast on complex event logs, as typically they are exponential. Therefore,

the α -algorithms are not very suitable to be applied to real-life logs.

Little Thumb (Weijters and van der Aalst 2003) extends the α algorithms with noise-handling capabilities: instead of considering binary activity relations, these relations are derived probabilistically and then filtered according to a user-set threshold.

Causal-Net Miners

The Flexible Heuristics Miner (FHM) (Weijters and Ribeiro 2011) uses the probabilistic activity relations of Little Thumb and focuses on soundness. To solve the issue of soundness, FHM returns causal nets, a model formalism in which it is defined that non-sound parts of the model are not part of the behaviour of the net.

The Fodina algorithm (vanden Broucke and Weerdt 2017) extends FHM with long-distance dependency support and, in some cases, duplicate activities. The Proximity miner (Yahya et al. 2016) extends FHM by incorporating domain knowledge. For more algorithms using causal nets, please refer to Weerdt et al. (2012) and Augusto et al. (2017b).

Even though causal nets are sound by definition, they place the burden of soundness checking on the interpreter/user of the net, and this still does not guarantee, for instance, that every activity in the model can be executed. Therefore, translating a causal net to a Petri net or BPMN model for further processing does not guarantee soundness of the translated model.

FHM, Fodina (<http://www.processmining.be/fodina>) and Proximity Miner (<https://sourceforge.net/projects/proxi-miner/>) are all available as ProM plug-ins and/or Apromore plug-ins.

Split Miner

To strike a different balance in log-quality criteria compared to IM that favours fitness, while improving in speed over ETM, Split Miner (SPM) (Augusto et al. 2017a) preprocesses the directly follows graph before constructing a BPMN model. In the preprocessing of directly follows graphs, first, loops and concurrency are identified and filtered out. Second, the graph is filtered in an optimisation step: each node must

be on a path from start to end, the total number of edges is minimised, while the sum of edge frequencies is maximised. Then, splits and joins (BPMN gateways) are inserted to construct a BPMN model.

SPM aims to improve over the precision of IM and the speed of ETM for real-life event logs. The balance between precision and fitness can be adjusted in the directly follows-optimisation step, which allows users to adjust the amount of noise filtering. However, the returned models are not guaranteed to be sound (proper completion is not guaranteed), and several OR-joins might be inserted, which increases complexity.

SPM is available as a plug-in of Apromore and as a stand-alone tool via <https://doi.org/10.6084/m9.figshare.5379190.v1>.

Conclusion

Many process mining techniques require a process model as a prerequisite. From an event log, process discovery algorithms aim to discover a process model, this model preferably having clear semantics, being sound, striking a user-adjustable balance between fitness, precision, generalisation and simplicity, and having confidence that the model represents the business process from which the event log was recorded. Three types of process discovery algorithms were discussed: directly follows-based techniques, soundness-guaranteeing algorithms and other algorithms, all targeting a subset of these quality criteria.

In explorative process mining projects, choosing a discovery algorithm and its parameters is a matter of repeatedly trying *soundness-guaranteeing* algorithms, evaluating their results using conformance checking and adjusting algorithm, parameters and event log as new questions pop up (van Eck et al. 2015).

Cross-References

- ▶ [Conformance Checking](#)
- ▶ [Decision Discovery in Business Processes](#)

- ▶ Event Log Cleaning for Business Process Analytics
- ▶ Process Model Repair

References

- Adriansyah A (2014) Aligning observed and modeled behavior. PhD thesis, Eindhoven University of Technology
- Augusto A, Conforti R, Dumas M, Rosa ML, Bruno G (2016) Automated discovery of structured process models: discover structured vs. discover and structure. In: Comyn-Wattiau I, Tanaka K, Song I, Yamamoto S, Saeki M (eds) Conceptual modeling – Proceedings of the 35th international conference, ER, Gifu, 14–17 Nov 2016. Lecture notes in computer science, vol 9974, pp 313–329. http://doi.org/10.1007/978-3-319-46397-1_25
- Augusto A, Conforti R, Dumas M, Rosa ML (2017a) Split miner: discovering accurate and simple business process models from event logs. In: IEEE international conference on data mining, New Orleans. <https://eprints.qut.edu.au/110153/>
- Augusto A, Conforti R, Dumas M, Rosa ML, Maggi FM, Marrella A, Mecella M, Soo A (2017b) Automated discovery of process models from event logs: review and benchmark. CoRR abs/1705.02288, <http://arxiv.org/abs/1705.02288>
- Badouel E (2012) On the α -reconstructibility of workflow nets. In: Haddad S, Pomello L (eds) Application and theory of Petri Nets – Proceedings of the 33rd international conference, PETRI NETS, Hamburg, 25–29 June 2012. Lecture notes in computer science, vol 7347. Springer, pp 128–147. http://doi.org/10.1007/978-3-642-31131-4_8
- Buijs JCAM, van Dongen BF, van der Aalst WMP (2012a) A genetic algorithm for discovering process trees. In: Proceedings of the IEEE congress on evolutionary computation, CEC, Brisbane, 10–15 June 2012. IEEE, pp 1–8. <http://doi.org/10.1109/CEC.2012.6256458>
- Buijs JCAM, van Dongen BF, van der Aalst WMP (2012b) On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman R, Panetto H, Dillon TS, Rinderle-Ma S, Dadam P, Zhou X, Pearson S, Ferscha A, Bergamaschi S, Cruz IF (eds) On the move to meaningful internet systems: OTM 2012, Proceedings of the confederated international conferences: CoopIS, DOA-SVI, and ODBASE, Rome, part I, 10–14 Sept 2012. Lecture notes in computer science, vol 7565. Springer, pp 305–322. http://doi.org/10.1007/978-3-642-33606-5_19
- Celonis (2017) Process mining. <https://www.celonis.com/>. [Online; Accessed 11 Nov 2017]
- DBL (2011) Proceedings of the IEEE symposium on computational intelligence and data mining, CIDM 2011, part of the IEEE symposium series on computational intelligence, 11–15 Apr 2011. IEEE, Paris. <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5937059>
- de Medeiros AKA, van Dongen BF, van der Aalst WMP, Weijters AJMM (2004) Process mining for ubiquitous mobile systems: an overview and a concrete algorithm. In: Baresi L, Dustdar S, Gall HC, Matera M (eds) Ubiquitous mobile information and collaboration systems, second CAiSE workshop, UMICS, Riga, 7–8 June 2004, Revised selected papers. Lecture notes in computer science, vol 3272. Springer, pp 151–165. http://doi.org/10.1007/978-3-540-30188-2_12
- Di Cicco C, Maggi FM, Mendling J (2016) Efficient discovery of target-branched declare constraints. Inf Syst 56:258–283. <http://doi.org/10.1016/j.is.2015.06.009>
- Ferilli S, Esposito F, Redavid D, Angelastro S (2016) Predicting process behavior in woman. In: Adorni G, Cagnoni S, Gori M, Maratea M (eds) AI*IA 2016: advances in artificial intelligence – Proceedings of the XVth international conference of the Italian association for artificial intelligence, Genova, 29 Nov–1 Dec 2016. Lecture notes in computer science, vol 10037. Springer, pp 308–320. http://doi.org/10.1007/978-3-319-49130-1_23
- Fluxicon (2017) Disco. <http://fluxicon.com>, [Online; Accessed 11 Nov 2017]
- Günther C, van der Aalst W (2007) Fuzzy mining-adaptive process simplification based on multi-perspective metrics. Business process management. Springer, Berlin/Heidelberg, pp 328–343
- Guo Q, Wen L, Wang J, Yan Z, Yu PS (2015) Mining invisible tasks in non-free-choice constructs. In: Motahari-Nezhad HR, Recker J, Weidlich M (eds) Business process management – Proceedings of the 13th international conference, BPM, Innsbruck, 31 Aug–3 Sept 2015. Lecture notes in computer science, vol 9253. Springer, pp 109–125. http://doi.org/10.1007/978-3-319-23063-4_7
- Leemans S (2017) Robust process mining with guarantees. PhD thesis, Technische Universiteit Eindhoven
- Leemans SJ, Fahland D, van der Aalst WMP (2013a) Discovering block-structured process models from event logs – a constructive approach. In: Colom JM, Desel J (eds) Application and theory of Petri Nets and concurrency – Proceedings of the 34th international conference, PETRI NETS, Milan, 24–28 June 2013. Lecture notes in computer science, vol 7927. Springer, pp 311–329. http://doi.org/10.1007/978-3-642-38697-8_17
- Leemans SJ, Fahland D, van der Aalst WMP (2013b) Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann N, Song M, Wohed P (eds) Business process management workshops – BPM 2013 international workshops, Beijing, 26 Aug 2013, Revised papers. Lecture notes in business information processing, vol 171. Springer, pp 66–78. http://doi.org/10.1007/978-3-319-06257-0_6
- Leemans SJ, Fahland D, van der Aalst WMP (2014a) Discovering block-structured process models from incomplete event logs. In: Ciardo G, Kindler E (eds)

- Application and theory of Petri Nets and concurrency – Proceedings of the 35th international conference, PETRI NETS, Tunis, 23–27 June 2014. Lecture notes in computer science, vol 8489. Springer, pp 91–110. http://doi.org/10.1007/978-3-319-07734-5_6
- Leemans SJJ, Fahland D, van der Aalst WMP (2014b) Process and deviation exploration with inductive visual miner. In: Limonad L, Weber B (eds) Proceedings of the BPM demo sessions 2014, co-located with the 12th international conference on business process management(BPM), Eindhoven, 10 Sept 2014, CEUR-WS.org, CEUR workshop proceedings, vol 1295, p 46. <http://ceur-ws.org/Vol-1295/paper19.pdf>
- Leemans SJJ, Fahland D, van der Aalst WMP (2015) Using life cycle information in process discovery. In: Reichert M, Reijers HA (eds) Business process management workshops – BPM, 13th international workshops, Innsbruck, 31 Aug–3 Sept 2015, Revised papers. Lecture notes in business information processing, vol 256. Springer, pp 204–217. http://doi.org/10.1007/978-3-319-42887-1_17
- Leemans SJJ, Fahland D, van der Aalst WMP (2016) Scalable process discovery and conformance checking. Softw Syst Model Special issue:1–33. <http://doi.org/10.1007/s10270-016-0545-x>
- Maggi FM, Mooij AJ, van der Aalst WMP (2011) User-guided discovery of declarative process models. In: DBL (2011), pp 192–199. <http://doi.org/10.1109/CIDM.2011.5949297>
- OMG (2011) Business process model and notation (BPMN) version 2.0. Technical report, Object management group (OMG)
- ProcessGold (2017) Enterprise platform. <http://processgold.com/en/>, [Online; Accessed 11 Nov 2017]
- Redlich D, Molka T, Gilani W, Blair GS, Rashid A (2014) Constructs competition miner: process control-flow discovery of bp-domain constructs. In: Sadiq SW, Soffer P, Völzer H (eds) Business process management – Proceedings of the 12th international conference, BPM, Haifa, 7–11 Sept 2014. Lecture notes in computer science, vol 8659. Springer, pp 134–150. http://doi.org/10.1007/978-3-319-10172-9_9
- Reisig W (1992) A primer in Petri net design. Springer compass international. Springer, Berlin/New York
- Rosa ML, Reijers HA, van der Aalst WMP, Dijkman RM, Mendling J, Dumas M, García-Bañuelos L (2011) APROMORE: an advanced process model repository. Expert Syst Appl 38(6):7029–7040. <http://doi.org/10.1016/j.eswa.2010.12.012>
- vanden Broucke SKLM, Weerdt JD (2017) Fodina: a robust and flexible heuristic process discovery technique. Decis Support Syst 100:109–118. <http://doi.org/10.1016/j.dss.2017.04.005>
- van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer <http://www.springer.com/gp/book/9783662498507>
- van der Aalst W, Weijters A, Maruster L (2004) Workflow mining: discovering process models from event logs. IEEE Trans Knowl Data Eng 16(9):1128–1142
- van der Werf JM, van Dongen BF, Hurkens CAJ, Serebrenik A (2009) Process discovery using integer linear programming. Fundam Inform 94(3–4):387–412. <http://doi.org/10.3233/FI-2009-136>
- van Dongen BF, de Medeiros AKA, Verbeek HMW, Weijters AJMM, van der Aalst WMP (2005) The prom framework: a new era in process mining tool support. In: Ciardo G, Daranodeau P (eds) Applications and theory of Petri Nets 2005, Proceedings of the 26th international conference, ICATPN, Miami, 20–25 June 2005. Lecture notes in computer science, vol 3536. Springer, pp 444–454. http://doi.org/10.1007/11494744_25
- van Eck ML, Lu X, Leemans SJJ, van der Aalst WMP (2015) PM'2 : a process mining project methodology. In: Zdravkovic J, Kirikova M, Johannesson P (eds) Advanced information systems engineering – Proceedings of the 27th international conference, CAiSE, Stockholm, 8–12 June 2015. Lecture notes in computer science, vol 9097. Springer, pp 297–313. http://doi.org/10.1007/978-3-319-19069-3_19
- van Zelst SJ, van Dongen BF, van der Aalst WMP, Verbeek HMW (2017) Discovering relaxed sound workflow nets using integer linear programming. CoRR abs/1703.06733. <http://arxiv.org/abs/1703.06733>
- Weerdt JD, Backer MD, Vanthienen J, Baesens B (2012) A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Inf Syst 37(7):654–676. <http://doi.org/10.1016/j.is.2012.02.004>
- Weijters AJMM, Ribeiro JTS (2011) Flexible heuristics miner (FHM). In: DBL (2011), pp 310–317. <http://doi.org/10.1109/CIDM.2011.5949453>
- Weijters AJMM, van der Aalst WMP (2003) Rediscovering workflow models from event-based data using little thumb. Integr Comput Aided Eng 10(2):151–162. <http://content.iospress.com/articles/integrated-computer-aided-engineering/ica00143>
- Wen L, Wang J, Sun J (2006) Detecting implicit dependencies between tasks from event logs. In: Zhou X, Li J, Shen HT, Kitsuregawa M, Zhang Y (eds) Frontiers of WWW research and development – APWeb 2006, Proceedings of the 8th Asia-Pacific web conference, Harbin, 16–18 Jan 2006. Lecture notes in computer science, vol 3841. Springer, pp 591–603. http://doi.org/10.1007/11610113_52
- Wen L, van der Aalst WMP, Wang J, Sun J (2007a) Mining process models with non-free-choice constructs. Data Min Knowl Discov 15(2):145–180. <http://doi.org/10.1007/s10618-007-0065-y>
- Wen L, Wang J, Sun J (2007b) Mining invisible tasks from event logs. In: Dong G, Lin X, Wang W, Yang Y, Yu JX (eds) Advances in data and web management, Joint 9th Asia-Pacific web conference, APWeb 2007, and Proceedings of the 8th international conference, on web-age information management, WAIM, Huang Shan, 16–18 June 2007. Lecture notes in computer science, vol 4505. Springer, pp 358–365. http://doi.org/10.1007/978-3-540-72524-4_38
- Wen L, Wang J, van der Aalst WMP, Huang B, Sun J (2009) A novel approach for process mining based on

- event types. *J Intell Inf Syst* 32(2):163–190. <http://doi.org/10.1007/s10844-007-0052-1>
- Wen L, Wang J, van der Aalst WMP, Huang B, Sun J (2010) Mining process models with prime invisible tasks. *Data Knowl Eng* 69(10):999–1021. <http://doi.org/10.1016/j.datak.2010.06.001>
- Yahya BN, Song M, Bae H, Sul S, Wu J (2016) Domain-driven actionable process model discovery. *Comput Ind Eng* 99:382–400. <http://doi.org/10.1016/j.cie.2016.05.010>

Automated Reasoning

Jeff Z. Pan¹ and Jianfeng Du²

¹University of Aberdeen, Scotland, UK

²Guangdong University of Foreign Studies,
Guangdong, China

Synonyms

Approximate reasoning; Inference; Logical reasoning; Semantic computing

Definitions

Reasoning is the process of deriving conclusions in a logical way. Automatic reasoning is concerned with the construction of computing systems that automate this process over some knowledge bases.

Automated Reasoning is often considered as a subfield of artificial intelligence. It is also studied in the fields of theoretical computer science and even philosophy.

Overview

The development of formal logic (Frege 1884) played a big role in the field of automated reasoning, which itself led to the development of artificial intelligence.

Historically, automated reasoning is largely related to theorem proving, general problem solvers, and expert systems (cf. the section

of “A Bit of History”). In the context of big data processing, automated reasoning is more relevant to modern knowledge representation languages, such as the W3C standard Web Ontology Language (OWL) (<https://www.w3.org/TR/owl2-overview/>), in which a knowledge base consists of a schema component (TBox) and a data component (ABox).

From the application perspective, perhaps the most well-known modern knowledge representation mechanism is Knowledge Graph (Pan et al. 2016b, 2017). In 2012, Google popularized the term “Knowledge Graph” by using it for improving its search engine. Knowledge Graphs are then adopted by most leading search engines (such as Bing and Baidu) and many leading IT companies (such as IBM and Facebook). The basic idea of Knowledge Graph is based on the knowledge representation formalism called semantic networks. There is a modern W3C standard for semantic networks called RDF (Resource Description Framework, <https://www.w3.org/TR/rdf11-concepts/>). Thus RDF/OWL graphs can be seen as exchangeable knowledge graphs, in the big data era.

While this entry will be mainly about automated reasoning techniques in the big data era, their classifications, key contributions, typical systems, as well as their applications, it starts with a brief introduction of the history.

A Bit of History

Many consider the Cornell Summer Meeting of 1957, which brought together many logicians and computer scientists, as the origin of automated reasoning.

The first automated reasoning systems were theorem provers, systems that represent axioms and statements in first-order logic and then use rules of logic, such as modus ponens, to infer new statements. The first system of this kind is the implementation of Presburger's decision procedure (which proved that the sum of two even numbers is even) by Davis (1957).

Another early type of automated reasoning system were general problem solvers, which at-

tempt to provide a generic planning engine that could represent and solve structured problems, by decomposing problems into smaller more manageable subproblems, solving each subproblem and assembling the partial answers into one final answer. The first system of this kind is Logic Theorist from Newell et al. (1957).

The first practical applications of automated reasoning were expert systems, which focused on much more well-defined domains than general problem solving, such as medical diagnosis or analyzing faults in an aircraft, and on more limited implementations of first-order logic, such as modus ponens implemented via IF-THEN rules. One of the forerunners of these systems is MYCIN by Shortliffe (1974).

Since 1980s, there have been prosperous studies of practical subsets of first-order logics as ontology languages, such as description logics (Baader et al. 2003) and answer set programming (Lifschitz 2002), as well as the standardization of ontology language OWL (version 1 in 2004 and version 2 in 2009). The wide adoption of ontology and Knowledge Graph (Pan et al. 2016b, 2017), including by Google and many other leading IT companies, confirms the status of ontology language in big data era.

In the rest of the entry, we will focus on automated reasoning with Ontology languages.

Classification

There can be different ways of classifying research problems related to automated ontology reasoning.

From the *purpose* point of view, automatic ontology reasoning can be classified into (1) deductive ontology reasoning (Levesque and Brachman 1987), which draws conclusions from given premises; (2) abductive ontology reasoning (Colucci et al. 2003), which finds explanations for observations that are not consequences of given premises; as well as (3) inductive ontology reasoning (Lisi and Malerba 2003), which concludes that all instances of a class have a certain property if some instances of the class have the property.

From the *direction* point of view, automatic ontology reasoning can be classified into (1) forward reasoning Baader et al. (2005), in which the inference starts with the premises, moves forward, and ends with the conclusions; (2) backward reasoning (Grosof et al. 2003), in which the inference starts with the conclusions, moves backward, and ends with the premises; as well as (3) bi-directional reasoning (MacGregor 1991) in which the inference starts with both the premises and the conclusions and moves forward and backward simultaneously or interactively, until the intermediate conclusions obtained by forward steps include all intermediate premises required by backward steps.

From the *monotonicity* point of view, automatic ontology reasoning can be classified into (1) monotonic ontology reasoning in which no existing conclusions will be dropped when new premises are added, as well as (2) nonmonotonic ontology reasoning (Quantz and Suska 1994) in which some existing conclusions can be dropped when new premises are added.

From the *scalability* point of view, automatic ontology reasoning can be classified into (1) parallel ontology reasoning (Bergmann and Quantz 1995), in which reasoning algorithms can exploit multiple computation cores in a computation nodes, and (2) distributed ontology reasoning (Borgida and Serafini 2003) and (Serafini 2005), in which reasoning algorithms can exploit a cluster of computation nodes. Scalable ontology reasoning is also often related to strategies of modularization (Suntisrivaraporn et al. 2008) and approximation (Pan and Thomas 2007).

From the *mobility* point of view, automated ontology reasoning can be classified into (1) reasoning with temporal ontologies (Artale and Franconi 1994), in which the target ontologies contain temporal constructors for class and property descriptions, and (2) stream ontology reasoning (Stuckenschmidt et al. 2010; Ren and Pan 2011), which, given some continuous updates of the ontology, requires updating reasoning results without naively recomputing all results.

From the *certainty* point of view, automatic reasoning can be classified into (1) ontology reasoning with certainty in which both premises and

conclusions are certain and either true or false, as well as (2) uncertainty ontology reasoning (Koller et al. 1997) in which either premises or conclusions are uncertain and often have truth values between 0/ – 1 and 1. There are different kinds of uncertainties within ontologies, such as probabilistic ontologies (Koller et al. 1997), fuzzy ontologies (Straccia 2001), and possibilistic ontologies (Qi et al. 2011).

Key Contributions

The highlight on contributions of automated ontology reasoning is the standardization of the Web Ontology Language (OWL).

The first version of OWL (or OWL 1) was standardized in 2004. It is based on the *SHOIQ* DL (Horrocks and Sattler 2005). However, there are some limitations of OWL 1:

1. The datatype support is limited (Pan and Horrocks 2006);
2. The only sub-language, OWL-Lite, of OWL 1 is not tractable;
3. The semantics of OWL 1 and RDF are not fully compatible (Pan and Horrocks 2003).

The second version of OWL (or OWL 2) was standardized in 2009. It is based on the *SROIQ* DL (Horrocks et al. 2006). On the one hand, OWL 2 has more expressive power, such as the stronger support of datatypes (Pan and Horrocks 2006; Motik and Horrocks 2008) and rules (Krötzsch et al. 2008). On the other hand, OWL 2 has three tractable sub-languages, including OWL 2 EL (Baader et al. 2005), OWL 2 QL (Calvanese et al. 2007), and OWL 2 RL (Grosof et al. 2003).

This two-layer architecture of OWL 2 allows approximating OWL 2 ontologies to those in its tractable sub-languages, such as approximations toward OWL 2 QL (Pan and Thomas 2007), toward OWL 2 EL (Ren et al. 2010), and toward OWL 2 RL (Zhou et al. 2013), so as to exploit efficient and scalable reasoners of the sub-languages. The motivation is based on the fact

that real-world knowledge and data are hardly perfect or completely digitalized.

Typical Reasoning Systems

Below are descriptions of some well-known OWL reasoners (in alphabetical order).

CEL

CEL (Baader et al. 2006) is a LISP-based reasoner for \mathcal{EL} (Baader et al. 2008), which covers the core part of OWL 2 EL. CEL is the first reasoner for the description logic \mathcal{EL} , supporting as its main reasoning task the computation of the subsumption hierarchy induced by \mathcal{EL} ontologies.

ELK

ELK (Kazakov et al. 2012) is an OWL 2 EL reasoner. At its core, ELK uses a highly optimized parallel algorithm (Kazakov et al. 2011). It supports stream reasoning in OWL 2 EL (Kazakov and Klinov 2013).

FaCT

FaCT Horrocks (1998) is a reasoner for the description logic *SHIF* (OWL-Lite). It is the first modern reasoner that demonstrates the feasibility of using optimized algorithms for subsumption checking in realistic applications.

FaCT++

FaCT++ (Tsarkov and Horrocks 2006) is a reasoner for (partially) OWL 2. It is the new generation of the well-known FaCT reasoner which is implemented using C++, with a different internal architecture and some new optimizations.

HermiT

HermiT (Glimm et al. 2014) is a reasoner for OWL 2. It is the first publicly available OWL 2 reasoner based on a hypertableau calculus (Motik et al. 2009), with a highly optimized algorithm for ontology classification (Glimm et al. 2010). HermiT can handle DL-Safe rules (Motik et al. 2005) on top of OWL 2.

Konclude

Konclude (Steigmiller et al. 2014b) is a reasoner for OWL 2. It supports almost all datatypes in OWL 2. Konclude implements a highly optimized version of tableau calculus enhanced with tableau saturation (Steigmiller and Glimm 2015). It supports parallel reasoning and nominal schemas (Steigmiller et al. 2014a) and DL-safe rules.

Mastro

Mastro (Calvanese et al. 2011) is an ontology-based data access (OBDA) management system for OWL 2 QL. It allows data to be managed by external relational data management or data federation systems. It uses the Presto algorithm Rosati and Almatelli (2010) for query rewriting.

Ontop

Ontop (Calvanese et al. 2016) is an ontology-based data access (OBDA) management system for RDF and OWL 2 QL, as well as SWRL with limited forms of recursions. It also supports efficient SPARQL-to-SQL mappings via R2RML (Rodriguez-Muro and Rezk 2015). Ontop has some optimizations on query rewriting based on database dependencies (Rodriguez-Muro et al. 2013).

Pellet

Pellet (Sirin et al. 2007) is a reasoner for OWL 2. It also has dedicated support for OWL 2 EL. It incorporates optimizations for nominals, conjunctive query answering, and incremental reasoning.

Racer

Racer (Haarslev and Möller 2001) is a reasoner for OWL 1. It has a highly optimized version of tableau calculus for the description logic $S\mathcal{H}\mathcal{T}\mathcal{Q}(D)$ (Horrocks and Patel-Schneider 2003).

RDFox

RDFox (Motik et al. 2014) is a highly scalable in-memory RDF triple store that supports shared memory parallel datalog (Ceri et al. 1989) reasoning. It supports stream reasoning (Motik et al.

2015b) and has optimizations for owl:sameAs (Motik et al. 2015a).

TrOWL

TrOWL (Thomas et al. 2010) is a highly optimized approximate reasoner (Pan et al. 2016a) for OWL 2. TrOWL outperforms some sound and complete reasoners in the time-constrained ORE (Ontology Reasoner Evaluation) competitions designed for sound and complete ontology reasoners. TrOWL has stream reasoning capabilities for both OWL 2 and OWL 2 EL (Ren and Pan 2011; Ren et al. 2016). It supports local closed world reasoning in NBox or closed predicates (Lutz et al. 2013).

Applications

Automated ontology reasoning has been widely used in web applications, such as for content management (BBC), travel planning and booking (Skyscanner), and web search (Google, Bing, Baidu).

It is also being applied in a growing number of vertical domains. One typical example is life science. For instance, OBO Foundry includes more than 100 biological and biomedical ontologies. The SNOMED CT (Clinical Terminology) ontology is widely used in healthcare systems of over 15 countries, including the USA, the UK, Australia, Canada, Denmark, and Spain. It is also used by major US providers, such as Kaiser Permanente. Other vertical domains include, but not limited to, agriculture, astronomy, oceanography, defense, education, energy management, geography, and geoscience.

While ontologies are widely used as structured vocabularies, providing integrated and user-centric view of heterogeneous data sources in the big data era, benefits of using automated ontology reasoning include:

1. Reasoning support is critical for development and maintenance of ontologies, in particular on derivation of taxonomy from class definitions and descriptions.

2. Easy location of relevant terms within large structured vocabulary;
3. Query answers enhanced by exploiting schema and class hierarchy.

An example in the big data context is the use of ontology and automated ontology reasoning for data access in Statoil, where about 900 geologists and geophysicists use data from previous operations in nearby locations to develop stratigraphic models of unexplored areas, involving diverse schemata and TBs of relational data spread over 1000s of tables and multiple databases. Data analysis is the most important factor for drilling success. 30–70% of these geologists and geophysicists' time is spent on data gathering. The use of ontologies and automated ontology reasoning enables better use of experts' time, reducing turnaround for new queries significantly.

Outlook

Despite the current success of automated ontology reasoning, there are still some pressing challenges in the big data era, such as the following:

1. Declarative data analytics (Kaminski et al. 2017) based on automated ontology reasoning;
2. Effective approaches of producing high-quality (Ren et al. 2014; Konev et al. 2014) ontologies and Knowledge Graphs;
3. Integration of automated ontology reasoning with data mining (Lecue and Pan. 2015) and machine learning (Chen et al. 2017) approaches.

References

- Artale A, Franconi E (1994) A computational account for a description logic of time and action. In: KR1994, pp 3–14
- Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (eds) (2003) The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York
- Baader F, Brandt S, Lutz C (2005) Pushing the EL envelope. In: IJCAI2005
- Baader F, Lutz C, Sunthiravaporn B (2006) CEL—a polynomial-time reasoner for life science ontologies. In: IJCAR'06, pp 287–291
- Baader F, Brandt S, Lutz C (2008) Pushing the EL envelope further. In: OWLED2008
- Bergmann F, Quantz J (1995) Parallelizing description logics. In: KI1995, pp 137–148
- Borgida A, Serafini L (2003) Distributed description logics. *J Data Semant* 1:153–184
- Calvanese D, Giacomo GD, Lembo D, Lenzerini M, Rosati R (2007) Tractable reasoning and efficient query answering in description logics: the dl-lite family. *J Autom Reason* 39:385–429
- Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Poggi A, Rodriguez-Muro M, Rosati R, Ruzzi M, Savo DF (2011) The MASTRO system for ontology-based data access. *J Web Semant* 2:43–53
- Calvanese D, Cogrel B, Komla-Ebri S, Kontchakov R, Lanti D, Rezk M, Rodriguez-Muro M, Xiao G (2016) Ontop: answering SPARQL queries over relational databases. *Semant Web J* 8:471–487
- Ceri S, Gottlob G, Tancar L (1989) What you always wanted to know about Datalog (and never dared to ask). *IEEE Trans Knowl Data Eng* 1:146–166
- Chen J, Lecue F, Pan JZ, Chen H (2017) Learning from ontology streams with semantic concept drift. In: IJCAI-2017, pp 957–963
- Colucci S, Noia TD, Sciascio ED, Donini F (2003) Concept abduction and contraction in description logics. In: DL2003
- Davis M (1957) A computer program for Presburgers' algorithm. In: Summaries of talks presented at the summer institute for symbolic logic, Cornell University, pp 215–233
- Frege G (1884) Die Grundlagen der Arithmetik. Breslau, Wilhelm Kobner
- Glimm B, Horrocks I, Motik B, Stoilos G (2010) Optimising ontology classification. In: ISWC2010, pp 225–240
- Glimm B, Horrocks I, Motik B, Stoilos G, Wang Z (2014) Hermit: an OWL 2 reasoner. *J Autom Reason* 53: 245–269
- Grosop BN, Horrocks I, Volz R, Decker S (2003) Description logic programs: combining logic programs with description logic. In: WWW2003, pp 48–57
- Haarslev V, Möller R (2001) RACER system description. In: IJCAR2001, pp 701–705
- Horrocks I (1998) Using an expressive description logic: FaCT or fiction? In: KR1998
- Horrocks I, Patel-Schneider P (2003) From SHIQ and RDF to OWL: the making of a web ontology language. *J Web Semant* 1:7–26
- Horrocks I, Sattler U (2005) A tableau decision procedure for shiq. In: IJCAI2005, pp 448–453
- Horrocks I, Kutz O, Sattler U (2006) The even more irresistible sroiq. In: KR2006, pp 57–67
- Kaminski M, Grau BC, Kostylev EV, Motik B, Horrocks I (2017) Foundations of declarative data analysis using limit datalog programs. In: Proceedings of the twenty-

- sixth international joint conference on artificial intelligence, IJCAI-17, pp 1123–1130. <https://doi.org/10.24963/ijcai.2017/156>
- Kazakov Y, Klinov P (2013) Incremental reasoning in OWL EL without bookkeeping. In: ISWC2013, pp 232–247
- Kazakov Y, Krötzsch M, Simancik F (2011) Concurrent classification of EL ontologies. In: ISWC2011, pp 305–320
- Kazakov Y, Krötzsch M, Simancik F (2012) ELK reasoner: architecture and evaluation. In: ORE2012
- Koller D, Levy A, Pfeffer A (1997) P-CLASSIC: a tractable probabilistic description logic. In: AAAI1997
- Konev B, Lutz C, Ozaki A, Wolter F (2014) Exact learning of lightweight description logic ontologies. In: KR2014, pp 298–307
- Krötzsch M, Rudolph S, Hitzler P (2008) Description logic rules. In: ECAI2008, pp 80–84
- Serafini L, Tamilin A (2005) Drago: distributed reasoning architecture for the semantic web. In: ESWC2005, pp 361–376
- Lecue F, Pan JZ (2015) Consistent knowledge discovery from evolving ontologies. In: AAAI-15
- Levesque HJ, Brachman RJ (1987) Expressiveness and tractability in knowledge representation and reasoning. *Comput Intell* 3:78–93
- Lifschitz V (2002) Answer set programming and plan generation. *Artif Intell* 138:39–54
- Lisi FA, Malerba D (2003) Ideal refinement of descriptions in AL-Log. In: ICILP2003
- Lutz C, Seylan I, Wolter F (2013) Ontology-based data access with closed predicates is inherently intractable (Sometimes). In: IJCAI2013, pp 1024–1030
- MacGregor RM (1991) Inside the LOOM description classifier. *ACM SIGART Bull – special issue on implemented knowledge representation and reasoning systems* 2:88–92
- Motik B, Horrocks I (2008) Owl datatypes: design and implementation. In: ISWC2008, pp 307–322
- Motik B, Sattler U, Studer R (2005) Query answering for OWL-DL with rules. *J Web Semant* 3:41–60
- Motik B, Shearer R, Horrocks I (2009) Hypertableau reasoning for description logics. *J Artif Intell Res* 36: 165–228
- Motik B, Nenov Y, Piro R, Horrocks I, Olteanu D (2014) Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In: AAAI2014
- Motik B, Nenov Y, Piro R, Horrocks I (2015a) Handling of owl: sameAs via rewriting. In: AAAI2015
- Motik B, Nenov Y, Robert Piro IH (2015b) Incremental update of datalog materialisation: the backward/forward algorithm. In: AAAI2015
- Newell A, Shaw C, Simon H (1957) Empirical explorations of the logic theory machine. In: Proceedings of the 1957 western joint computer conference
- Pan JZ, Horrocks I (2003) RDFS(FA) and RDF MT: two semantics for RDFS. In: Fensel D, Sycara K, Mylopoulos J (eds) ISWC2003
- Pan JZ, Horrocks I (2006) OWL-EU: adding customised datatypes into OWL. *J Web Semant* 4: 29–39
- Pan JZ, Thomas E (2007) Approximating OWL-DL ontologies. In: The proceedings of the 22nd national conference on artificial intelligence (AAAI-07), pp 1434–1439
- Pan JZ, Ren Y, Zhao Y (2016a) Tractable approximate deduction for OWL. *Artif Intell* 235:95–155
- Pan JZ, Vetere G, Gomez-Perez J, Wu H (2016b) Exploiting linked data and knowledge graphs for large organisations. Springer, Switzerland
- Pan JZ, Calvanese D, Eiter T, Horrocks I, Kifer M, Lin F, Zhao Y (2017) Reasoning web: logical foundation of knowledge graph construction and querying answering. Springer, Cham
- Qi G, Ji Q, Pan JZ, Du J (2011) Extending description logics with uncertainty reasoning in possibilistic logic. *Int J Intell Syst* 26:353–381
- Quantz JJ, Suska S (1994) Weighted defaults in description logics: formal properties and proof theory. In: Annual conference on artificial intelligence, pp 178–189
- Ren Y, Pan JZ (2011) Optimising ontology stream reasoning with truth maintenance system. In: CIKM 2011
- Ren Y, Pan JZ, Zhao Y (2010) Soundness preserving approximation for TBox reasoning. In: AAAI2010
- Ren Y, Parvizi A, Mellish C, Pan JZ, van Deemter K, Stevens R (2014) Towards competency question-driven ontology authoring. In: Proceedings of the 11th conference on extended semantic web conference (ESWC 2014)
- Ren Y, Pan JZ, Guclu I, Kollingbaum M (2016) A combined approach to incremental reasoning for EL ontologies. In: RR2016
- Rodriguez-Muro M, Rezk M (2015) Efficient SPARQL-to-SQL with R2RML mappings. *J Web Semant* 33:141–169
- Rodriguez-Muro M, Kontchakov R, Zakharyashev M (2013) Query rewriting and optimisation with database dependencies in ontop. In: DL2013
- Rosati R, Almatelli A (2010) Improving query answering over DL-Lite ontologies. In: KR2010, pp 290–300
- Shortliffe EH (1974) MYCIN: a rule-based computer program from advising physicians regarding antimicrobial therapy selection. PhD thesis, Stanford University
- Sirin E, Parsia B, Grau B, Kalyanpur A, Katz Y (2007) Pellet: a practical owl-dl reasoner. *J Web Semant* 5:51–53
- Steigmiller A, Glimm B (2015) Pay-As-You-Go description logic reasoning by coupling tableau and saturation procedure. *J Artif Intell Res* 54:535–592
- Steigmiller A, Glimm B, Liebig T (2014a) Reasoning with nominal schemas through absorption. *J Autom Reason* 53:351–405
- Steigmiller A, Liebig T, Glimm B (2014b) Konclude: system description. *J Web Semant* 27:78–85
- Straccia U (2001) Reasoning within fuzzy description logics. *J Artif Intell Res* 14:147–176

- Stuckenschmidt H, Ceri S, Valle ED, van Harmelen F (2010) Towards expressive stream reasoning. In: Semantic challenges in sensor networks, no. 10042 in Dagstuhl seminar proceedings
- Suntisrivaraporn B, Qi G, Ji Q, Haase P (2008) A modularization-based approach to finding all justifications for OWL DL entailments. In: ASWC2008, pp 1–15
- Thomas E, Pan JZ, Ren Y (2010) TrOWL: tractable OWL 2 reasoning infrastructure. In: ESWC2010
- Tsarkov D, Horrocks I (2006) FaCT++ description logic reasoner: system description. In: IJCAR2006, pp 292–297
- Zhou Y, Grau BC, Horrocks I, Wu Z, Banerjee J (2013) Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In: WWW2013, pp 1569–1580

Availability

► [Security and Privacy in Big Data Environment](#)

B

Benchmark Harness

Nicolas Michael
Oracle, San Francisco, CA, USA

Synonyms

Test harness

Definitions

A benchmark harness is a software that provides the infrastructure to conduct benchmarks of a software and/or hardware system, typically with the goal to quantitatively assess the system's characteristics and capabilities or to compare the characteristics and capabilities of multiple systems relative to each other. It facilitates the development and execution of benchmarks and the analysis of benchmark results. Typical components of a big data benchmark harness include a tool to generate data, an execution environment to run benchmarks, a data collection component to monitor the benchmark run, and a reporting component to calculate and summarize benchmark results.

Overview

In computing, benchmarking is the process of assessing a system's quantitative characteristics

and capabilities by running a benchmark workload (or set of workloads) against it. The assessed system, also referred to as the *system under test (SUT)*, may be a software and/or hardware system. The SUT's characteristics and capabilities of interest are often its performance but may also include its availability, reliability, quality-of-service, cost, or other nonfunctional aspects. A benchmark workload exercises functionality of the SUT by submitting requests or jobs to the system, invoking its operations, or through other triggers, typically in a way representative for a common use of the system.

The primary objective of the benchmark workload is to measure the SUT's operations in a quantitative way using metrics such as throughput, capacity, execution time, or resource usage. The validation for correctness of SUT's responses or outputs, however, is often of lower priority, and correct operation is assumed as a prerequisite. Benchmark results are typically expressed as a set of metrics, which can be used to compare the performance or other nonfunctional aspects of different systems, or to optimize or tune a system. A *benchmark harness* facilitates this process by providing the necessary infrastructure to conduct benchmarks.

While the terms *benchmarking* and *testing* and consequently *benchmark harness* and *test harness* are often used interchangeably, the term *benchmarking* always refers to a quantitative evaluation of a system's nonfunctional operation such as its performance, while the term *testing*

often refers to a functional validation of a system's specific behavior.

A benchmark harness often provides reusable functionality that can be shared across a set of benchmarks or workloads and provides well-defined interfaces to implement or integrate multiple benchmarks or workloads into the same harness and to execute them. Such common functionality may include:

- Preparing and configuring the test environment
- Generating and loading of data
- Starting, supervising, and stopping of a benchmark workload
- Scheduling and accounting of benchmark operations
- Monitoring of the system under test
- Collecting runtime statistics from the benchmark workload and system under test
- Generating benchmark reports (including benchmark metrics and result validation)

Benchmark harnesses typically implement a subset, if not all, of the above features, which are discussed in more detail below.

In the context of big data, the generation of test data is a crucial part of many big data benchmarks. Real-world data sets are often difficult to obtain and not necessarily of the desired scale. Many big data benchmarks therefore rely on either synthetic data sets or synthetically scaled-up real-world data sets. The ability to generate meaningful test data at any scale, suitable for reproducible benchmark runs, is therefore important for most big data benchmark harnesses.

Key Research Findings

While there is no single architecture for benchmark harnesses, many benchmark harnesses are structured in a similar way and provide some or all of the above-noted features. Research around benchmark harnesses is sparse, and only little detailed description of benchmark harness features

exist in literature. This article summarizes some of them and provides an overview of features and functionality a benchmark harness should provide.

Han (Han et al. 2014) formulates requirements on how to design big data benchmarks (or big data benchmark harnesses). He proposes a design consisting of a user interface layer, a function layer, and an execution layer. The user interface layer shall provide users the ability to select benchmark workloads, data sets, and scale and to initiate benchmark runs. The function layer shall consist of data generators, test generators, and metrics, while the execution layer shall provide system configuration tools, data format conversion tools, a result analyzer, and a result reporter. All these components shall be shared across benchmarks and data sets and provide mechanisms to implement various benchmark workloads on a common harness, which can be adapted to a variety of environments.

Test Environment Preparation

Before a benchmark can be run, the test environment needs to be prepared and configured. Depending on the complexity of the test environment, its degree of automation or integration with deployment tools, the preparation of a test environment can be a lengthy, tedious, and error-prone task. Han therefore proposes that benchmark harnesses should have “configuration tools” to enable a benchmark to run on a specific software stack (Han et al. 2014). Such tools should help a user to set up a properly and compliantly configured test environment.

Since most benchmarks are designed to be executed against arbitrary systems of different vendors, which might evolve over time or even are released only after the benchmark has been created, it is generally difficult for a benchmark harness to integrate with the software and hardware stack of every single relevant product. In some cases, benchmark harnesses may provide configuration tools for selective products. Ideally, benchmark harnesses should be extensible by providing interfaces that allow users to integrate them with various products.

The product-agnostic benchmark harness CloudPerf (Michael et al. 2017) provides APIs for additional “service modules” to implement “precondition checks” that validate a system configuration before a test run and potentially take corrective action. Each service may also implement a preparation method to set up or prepare a test environment, which is invoked through a callback before the workload starts running.

Test Data Generation

Test data is a central aspect of every big data benchmark. Most big data benchmarks rely on synthetically generated or synthetically scaled-up real-world data sets. Baru argues that the pure use of real data sets is often impractical as those can be hard to obtain, download, and store but most importantly are difficult to scale (Baru et al. 2012). As Han states “synthetic data generation preserving the 4V properties of big data is the foundation of producing meaningful and credible evaluation results”(Han et al. 2014). Rabl discusses requirements for parallel data generation and introduces the parallel data generation framework PDGF (Rabl and Poess 2011). Wang describes approaches and trade-offs of synthetic data generation (Wang et al. 2014), and many other researchers have written about this subject (Baru et al. 2012; Ghazal et al. 2013).

Han describes an architecture for test data generators in benchmark harnesses, including format conversion tools as a post-processing step to convert the generated data into a format usable by a specific workload and environment. BigDataBench uses BDGS as a data generator (Ming et al. 2013). Capotă et al. describe data set generation as one module of the Graphalytics architecture (Capotă et al. 2015).

Benchmark Workload Execution

The central function of every benchmark harness is the execution of the benchmark workload itself. As a minimum, the benchmark harness must support features to start and stop the benchmark workload or to detect its completion. Typically the harness would also monitor and supervise the benchmark workload while it is running.

Some benchmark harnesses provide additional features for workload execution through abstractions or APIs. Graphalytics has a platform-specific algorithm implementation to adapt benchmark functions to particular big data platforms (Capotă et al. 2015). CloudPerf *drivers*, which are started and supervised by the CloudPerf framework, can each execute an individual workload and either wrap existing workloads or run native workloads implemented using CloudPerf’s workload modeling APIs (Michael et al. 2017).

Scheduling and Accounting of Benchmark Operations

A big data benchmark typically consists of multiple operations which are executed either randomly or in a particular order at a configurable concurrency, rate, or timing. The elapsed time of benchmark operations as well as the total number of successfully completed operations, number of failures, or rate and concurrency of operations must be captured during workload execution to calculate benchmark metrics and evaluate the benchmark run after completion.

Most benchmark harnesses rely on the benchmark workload itself to schedule and monitor its benchmark operations and only gather the collected workload statistics through some defined interface such as a log file. In this model, each benchmark workload might use different methodology to schedule operations and account for them. Alternatively, a benchmark harness could have the ability to schedule and account for benchmark operations directly built into the harness. Benchmark workloads would then only implement the benchmark operations itself, while the infrastructure to run and measure operations is provided by a common harness. The benchmark harness must then specify interfaces for benchmark workloads to implement benchmark operations, which are then invoked by the harness during a benchmark run. CloudPerf follows this models for natively implemented workloads (Michael et al. 2017), while BigDataBench relies on each individual benchmark to capture schedule and account for its operations (Wang et al. 2014).

System Monitoring and Statistics Collection

The previous section discussed the need to monitor the benchmark workload itself. The objective of running a benchmark can be technical or competitive (Baru et al. 2012). In the technical use of benchmarks, the objective is to optimize a system, while in the competitive use of a benchmark, the objective is to compare the relative performance of different systems. Both cases require the collection of additional system statistics to fully evaluate a benchmark run, analyze its resource usage, and eventually optimize the system.

A benchmark harness should therefore also be capable of collecting system statistics across all relevant layers of the stack during a benchmark run. In the Graphalytics benchmark harness, a “system monitor” component is “responsible for gathering resource utilization statistics from the SUT” (Capotă et al. 2015). CloudPerf’s statistics framework enables application- and environment-specific statistics providers to capture and push statistics into the harness for live monitoring and post-run analysis (Michael et al. 2017).

Benchmark Report Generation

The outcome of a benchmark run is typically documented in a report. The benchmark report should include information about the benchmark itself, its configuration and scale, and the system environment on which it was conducted and contain benchmark metrics as well as relevant system statistics. Its intention is to summarize the outcome of a benchmark and potentially validate its result to classify it as “passed” or “failed”. The report should allow to compare two different systems against each other and allow to judge them with respect to their relative performance. Additional desirable metrics include each system’s cost (in terms of necessary system components or power consumption), quality-of-service, and maximum capacity with respect to data volume, supported users, or concurrency. For technical benchmarking (Ghazal et al. 2013), the report should further include relevant system

statistics that help engineers to analyze and troubleshoot performance bottlenecks and tune or optimize a system.

Han describes components for their layered harness architecture that allow the definition of metrics and the analysis and reporting of results (Han et al. 2014). Metrics can be classified as user-perceived metrics such as request rates and response times and architectural metrics such as resource utilization and other system statistics from the SUT (Wang et al. 2014).

Graphalytics (Capotă et al. 2015) contains an output validator module, which receives data from the benchmark workload itself as well as the big data platform. Benchmark configuration data from the benchmark, evaluation of the benchmark’s correctness from the output validator, as well as system statistics from the system monitor component are then processed by the report generator to compile a benchmark report, which is then stored in a result database. CloudPerf (Michael et al. 2017) supports the definition of metrics, result checks, and key performance indicators (KPIs), which are evaluated by a reporter component at the end of a run based on both workload and system statistics. The report includes system configuration information, benchmark and system metrics, KPIs, a validation whether the benchmark run was successful or failed based on the configured result checks, as well as detailed workload and system runtime statistics.

Examples of Application

BigDataBench (Wang et al. 2014) is a benchmark suite of a rather loosely coupled collection of benchmarks, data sets, and data generators (Ming et al. 2013), with little common harness functionality across the various benchmarks, but contains a large number of benchmark workloads (originally 19, which have since then been enhanced to 37 benchmark workloads (Zhan et al. 2016)). Graphalytics on the other hand implements a modular and extensible benchmark harness “which facilitates the addition of new

datasets, algorithms, and platforms, as well as the actual performance evaluation process, including result collection and reporting” (Capotă et al. 2015). CloudPerf (Michael et al. 2017), a benchmark harness primarily designed for distributed multi-tenant systems, consists of an extensible framework with well-defined APIs to implement workloads, data loading, statistics collection, fault injection, and cloud deployment while providing common functionality in the harness itself.

Cross-References

► [System Under Test](#)

References

- Baru C, Bhandarkar M, Nambiar R, Poess M, Rabl T (2012) Setting the direction for big data benchmark standards. In: Technology conference on performance evaluation and benchmarking. Springer, Berlin, pp 197–208
- Capotă M, et al (2015) Graphalytics: a big data benchmark for graph-processing platforms. Proceedings of the GRADES’15. ACM, New York
- Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen HA (2013) BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. ACM, New York, pp 1197–1208
- Han R, Lu X, Jiangtao X (2014) On big data benchmarking, Workshop on big data benchmarks, performance optimization, and emerging hardware. Springer, Cham
- Michael N, et al (2017) CloudPerf: a performance test framework for distributed and dynamic multi-tenant environments. Proceedings of the 8th ACM/SPEC on international conference on performance engineering. ACM, New York
- Ming Z et al (2013) BDGS: a scalable big data generator suite in big data benchmarking, Workshop on big data benchmarks. Springer, Cham
- Rabl T, Poess M (2011) Parallel data generation for performance analysis of large, complex RDBMS. In: Proceedings of the fourth international workshop on testing database systems. ACM, New York, p 5
- Wang L, et al (2014) Bigdatabench: a big data benchmark suite from internet services. High Performance Computer Architecture (HPCA), 2014 IEEE 20th international symposium on. IEEE, New York

Zhan J, et al (2016) BigDataBench tutorial. ASPLOS. http://www.bafst.com/events/asplos16/BigDataBench_tutorial_16/wp-content/uploads/BigDataBench_First_Part-asplos16.pdf

B

Big Data Analysis and IoT

Yongrui Qin¹ and Quan Z. Sheng²

¹School of Computing and Engineering,
University of Huddersfield, Huddersfield, UK

²Department of Computing, Macquarie
University, Sydney, NSW, Australia

Definitions

Intrinsic characteristics of IoT big data are identified and classified. Potential IoT applications bringing significant changes in many domains are described. Many new challenges and open issues to be addressed in IoT are discussed.

Overview

This entry firstly discusses the intrinsic characteristics of IoT big data and classifies them into three categories: data generation, data quality, and data interoperability. Specific characteristics of each category are identified. Then it introduces ongoing and/or potential IoT applications, which show that IoT can bring significant changes in many domains, i.e., cities and homes, environment monitoring, health, energy, business, etc. Since many new challenges and issues around IoT big data have not been addressed, which require substantial efforts from both academia and industry, this entry also identifies several key directions for future research and development from a data-centric perspective.

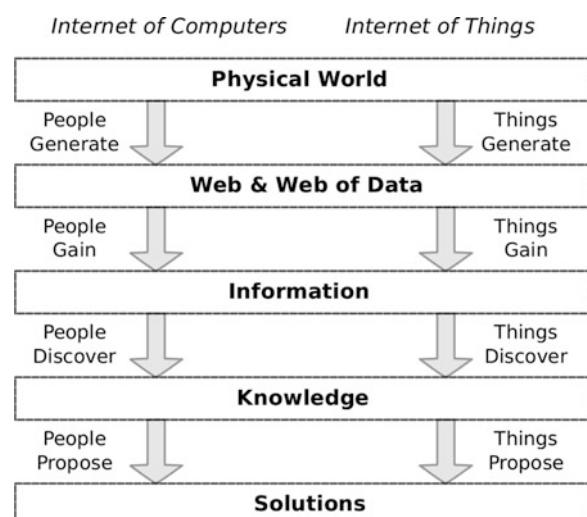
Introduction

The Internet is a global system of networks that interconnect computers using the standard Internet protocol suite. It has significant impact on the world as it can serve billions of users worldwide. Millions of private, public, academic, business, and government networks, of local to global scope, all contribute to the formation of the Internet. The traditional Internet has a focus on computers and can be called the Internet of Computers. In contrast, evolving from the Internet of Computers, the Internet of Things (IoT) emphasizes things rather than computers (Ashton 2009). It aims to connect everyday objects, such as coats, shoes, watches, ovens, washing machines, bikes, cars, and even humans, plants, animals, and changing environments, to the Internet to enable communication/interactions between these objects. The ultimate goal of IoT is to enable computers to see, hear, and sense the real world. It is predicted by Ericsson that the number of Internet-connected things will reach 50 billion by 2020. Electronic devices and systems exist around us providing different services to the people in different situations: at home, at work, in their office, or driving a car on the street. IoT also enables the close relationship between human and opportunistic connection of smart things (Guo et al. 2013).

Big Data Analysis and IoT, Fig. 1 Internet of Computers v.s. Internet of Things

There are several definitions or visions of IoT from different perspectives. From the viewpoint of services provided by things, IoT means “a world where things can automatically communicate to computers and each other providing services to the benefit of the human kind” (CASAGRAS 2000). From the viewpoint of connectivity, IoT means “from anytime, anyplace connectivity for anyone, we will now have connectivity for anything” (ITU 2005). From the viewpoint of communication, IoT refers to “a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols” (INFSO 2008). Finally, from the viewpoint of networking, IoT is the Internet evolved “from a network of interconnected computers to a network of interconnected objects” (European Commission 2009).

We focus on our study of IoT from a *data perspective*. As shown in Fig. 1, data is processed differently in the Internet of Things and traditional Internet environments (i.e., Internet of Computers). In the Internet of Computers, both main data producers and consumers are human beings. However, in the Internet of Things, the main actors become *things*, which means things are the majority of data producers and consumers. Therefore, we give our definition of IoT as follows:



In the context of the Internet, addressable and interconnected things, instead of humans, act as the main data producers, as well as the main data consumers. Computers will be able to learn and gain information and knowledge to solve real world problems directly with the data fed from things. As an ultimate goal, computers enabled by the Internet of Things technologies will be able to sense and react to the real world for humans.

As of 2012, 2.5 quintillion (2.5×10^{18}) bytes of data are created daily (<http://www-01.ibm.com/software/data/bigdata/>). In IoT, connecting all of the things that people care about in the world becomes possible. All these things would be able to produce much more data than nowadays. The volumes of data are vast, the generation speed of data is fast, and the data/information space is global (James et al. 2009). Indeed, IoT is one of the major driving forces for *big data analytics*. Given the scale of IoT, topics such as storage, distributed processing, real-time data stream analytics, and event processing are all critical, and we may need to revisit these areas to improve upon existing technologies for applications of this scale.

In this entry, we systematically investigate the key technologies related to the development of IoT applications, particularly from a data-centric perspective. Our aim is to provide a better understanding of IoT data characteristics, applications, and issues. Although some reviews about IoT have been conducted recently (e.g., Atzori et al. 2010; Zeng et al. 2011; An et al. 2013; Perera et al. 2013; Li et al. 2014; Yan et al. 2014), they focus on high-level general issues and are mostly fragmented. Sheng et al. (2017) covers a number of areas in Web of Things. These efforts do not specifically cover topics on IoT data characteristics and potential applications from a data-centric perspective, which is fundamentally critical to fully embrace IoT.

The remainder of the chapter is organized as follows. Section “**IoT Data Taxonomy**” identifies an IoT data taxonomy. In section “**Potential IoT Applications**”, some typical ongoing and/or potential IoT applications where data techniques for IoT can bring significant changes are described. Finally, section “**Open Issues**” highlights

some research open issues on IoT from the data perspective, and section “**Summary**” offers some concluding remarks.

B

IoT Data Taxonomy

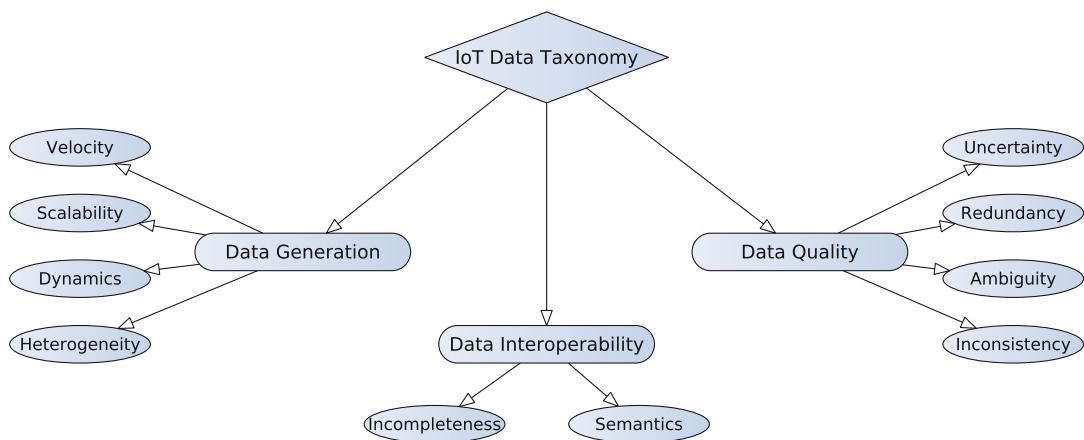
In this section, we identify the intrinsic characteristics of IoT data and classify them into three categories: *data generation*, *data quality*, and *data interoperability*. We also identify specific characteristics of each category, and the overall IoT data taxonomy is shown in Fig. 2.

Data Generation

- **Velocity.** In IoT, data can be generated at different rates. For example, for GPS-enabled moving vehicles in road networks, the GPS signal sampling frequency could be every few seconds, every few minutes, or even every half an hour. But some sensors can scan at a rate up to 1,000,000 sensing elements per second (<https://www.tekscan.com/support/faqs/what-are-sensors-sampling-rates>). On one hand, it is challenging to handle very high sampling rates, which require efficient processing of the fast generated data. On the other hand, it is also challenging to deal with low sampling rates, due to the fact that some important information may be lost for data processing and decision making.

- **Scalability.** Since things are able to continuously generate data together with the foreseeable excessively large number of things, the IoT data is expected to be at an extremely large scale. It is easy to image that, in IoT data processing systems, scalability will be a long-standing issue, aligning with the current big data trend.

- **Dynamics.** There are many dynamic elements within IoT data. Firstly, many things are mobile, which will lead to different locations at different times. Since they will move to different environments, the sensing results of things will also be changing to reflect the real world. Secondly, many things are fragile. This means the generated data will change over time due to the failure of things. Thirdly, the



Big Data Analysis and IoT, Fig. 2 IoT data taxonomy

connections between things could be intermittent. This also creates dynamics in any IoT data processing system.

- **Heterogeneity.** There will be many kinds of things potentially connecting to the Internet in the future, ranging from cars, robots, fridges, and mobile phones to shoes, plants, watches, and so on. These kinds of things will generate data in different formats using different vocabularies. In addition, there will be assorted IoT data processing systems, which will also provide data in customized formats to tailor different data needs.

Data Quality

- **Uncertainty.** In IoT, uncertainty may come from different sources. In RFID data, the uncertainty can refer to missing readings, readings of nonexisting IDs, etc. In wireless sensor networks, uncertainty can refer to sensing precision (the degree of reproducibility of a measurement), accuracy (the maximum difference that will exist between the actual value and the indicated value), etc.
- **Redundancy.** Redundancy can also be easily observable in IoT. For example, in RFID data, the same tags can be read multiple times at the same location (because multiple RFID readers exist at the spot or tags are read multiple times at the same spot) or at different locations. In wireless sensor networks, a group

of sensors of the same type may also be deployed in a nearby area, which can produce similar sensing results of that area. For the same sensor, due to the possible high sampling rates, redundant sensing data can be produced.

- **Ambiguity.** Dealing with large amount of ambiguity in IoT data is inevitable. The data produced by assorted things can be interpreted in different ways due to different data needs from different things or other data consumers. Such data can also be useful and important to any other things, which brings about the challenges of proper interpretation of the produced data to different data consumers.

- **Inconsistency.** Inconsistency is also prevalent in IoT data. For example, in RFID data, inconsistency can occur due to missing readings of tags at some locations along the supply chain. It is also easy to observe inconsistency in sensing data as when multiple sensors are monitoring the same environment and reporting sensing results. Due to the precision and accuracy of the sensing process and other problems including packet loss during transmission, data inconsistency is also an intrinsic characteristic in sensing data.

Data Interoperability

- **Incompleteness.** In order to process IoT data, being able to detect and react to events in real

time, it is important to combine data from different types of data sources to build a big and complete picture of relevant backgrounds of the real world. However, as this process relies on the cooperation of mobile and distributed things that are generating relevant background data, incompleteness is easily observable in IoT data. Suppose there are a large number of available data sources. It is of great importance to determine which data sources can best address the incompleteness of data for a given data processing task.

- **Semantics.** To address the challenges posed by the deluge of IoT data, things, or machines acting as the major data consumers should be a promising trend for data processing in the IoT era. Inspired by Semantic Web technologies, in order to enable machines to understand data for human beings, injecting semantics into data could be an initial step. Therefore, semantics within IoT data will play an important role in the process of enabling things/machines to understand and process IoT data by themselves.

Potential IoT Applications

As pointed out by Ashton (2009) that IoT “has the potential to change the world, just as the Internet did.” The ongoing and/or potential IoT applications show that IoT can bring significant changes in many domains, i.e., cities and homes, environment monitoring, health, energy, business, etc. IoT can bring the ability to react to events in the physical world in an automatic, rapid, and informed manner. This also opens up new opportunities for dealing with complex or critical situations and enables a wide variety of business processes to be optimized. In this section, we overview several representative domains where IoT can make some profound changes.

Smart Cities and Homes

IoT can connect billions of smart things and can help capture information in cities. Based on IoT, cities would become smarter and more efficient. Below are some examples of promising IoT

applications in future smart cities. In a modern city, lots of digital data traces are generated there every second via cameras and sensors of all kinds (Guinard 2010). All this data represents a gold mine for everyone, if people in the city would be able to take advantage of it in an efficient and effective way. For example, IoT can facilitate resource management issues for modern cities. Specifically, static resources (e.g., fire stations, parking spots) and mobile resources (e.g., police cars, fire trucks) in a city can be managed effectively using IoT technologies. Whenever events (fires, crime reports, cars looking for parking) arise, IoT technologies would be able to quickly match resources with events in an optimal way based on the information captured by smart things, thereby reducing cost and saving time. Taxi drivers in the city would also be able to better serve prospective passengers by learning passenger’s mobility patterns and other taxi drivers’ serving behaviors through the help of IoT technologies (Yuan et al. 2011). One study estimated a loss of \$78 billion in 2007 in the form of 4.2 billion lost hours and 2.9 billion gallons of wasted gasoline in the United States alone (Mathur et al. 2010). IoT could bring fundamental changes in urban street parking management, which would greatly benefit the whole society by reducing traffic congestion and fuel consumption.

Security in a city is of great concerns, which can benefit a lot from the development of IoT technologies. Losses resulted from property crimes were estimated to be \$17.2 billion in the United States in 2008 (Guha et al. 2010). Current security cameras, motion detectors, and alarm systems are not able to help track or recover stolen property. IoT technologies can help to deter, detect, and track personal property theft since things are interconnected and can interact with each other. IoT technologies can also help improve stolen property recovery rates and disrupt stolen property distribution networks. Similarly, a network of static and mobile sensors can be used to detect threats on city streets and in open areas such as parks.

With IoT technologies, people can browse and manage their homes via the Web. For example,

they would be able to check whether the light in their bedrooms is on and could turn it off by simply clicking a button on a Web page. Similar operations and management could be done in office environments. Plumbing is ranked as one of the ten most frequently found problems in homes (Lai et al. 2010). It is important to determine the spatial topology of hidden water pipelines behind walls and underground. In IoT, smart things in homes would be able to report plumbing problems automatically and report to owners and/or plumbers for efficient maintenance and repair.

Environment Monitoring

IoT technologies can also help to monitor and protect environments thereby improving human's knowledge about environments. Take water as an example. Understanding the dynamics of bodies of water and their impact on the global environment requires sensing information over the full volume of water. In such context, IoT technologies would be able to provide effective approaches to study water. Also IoT could improve water management in a city. Drinking water is becoming a scarce resource around the world. In big cities, efficiently distributing water is one of the major issues (Guinard 2010). Various reports show that on average 30% of drinkable water is lost during transmission due to the aging infrastructure and pipe failures. Further, water can be contaminated biologically or chemically due to inefficient operation and management. In order to effectively manage and efficiently transport water, IoT technologies would be of great importance.

Soil contains vast ecosystems that play a key role in the Earth's water and nutrient cycles, but scientists cannot currently collect the high-resolution data required to fully understand them. Many soil sensors are inherently fragile and often produce invalid or uncalibrated data (Ramanathan et al. 2009). IoT technologies would help to validate, calibrate, repair, or replace sensors, allowing to use available sensors without sacrificing data integrity and meanwhile minimizing the human resources required.

Sound is another example where IoT technologies can help. Sound is multidimensional,

varying in intensity and spectra. So it is difficult to quantify, e.g., it is difficult to determine what kind of sound is noise. Further, the definitions and feelings of noise are quite subjective. For example, some noises could be pleasant, like flowing water, while others can be annoying, such as car alarms, screeching breaks, and people arguing. A device has been designed and built to monitor residential noise pollution to address the above problems (Zimmerman and Robson 2011). Firstly, noise samples from three representative houses are used, which span the spectrum of quiet to noisy neighborhoods. Secondly, a noise model is developed to characterize residential noise. Thirdly, noise events of an entire day (24 h) are compressed into a 1-minute auditory summary. Data collection, transmission, and storage requirements can be minimized in order to utilize low-cost and low-power components, while sufficient measurement accuracy is still maintained.

Intel has developed smart sensors that can warn people about running outside when the air is polluted (<http://www.fastcoexist.com/1680111/intels-sensors-will-warn-you-about-running-outsides-when-the-air-is-polluted>). For example, if someone is preparing to take a jog along his/her regular route, an application on his/her smartphone pushes out a message: air pollution levels are high in the park where he/she usually runs. Then he/she could try a recommended route that is cleaner. Currently, many cities already have pollution and weather sensors. They are usually located on top of buildings, far from daily human activities.

Health

Covering a number of markets including manufacturing, transportation, systems, software, services, and so on, IoT in healthcare is expected to grow to a market size of \$300 billion by 2022 (Firouzi et al. 2018). In future IoT environments, an RFID-enabled information infrastructure would be likely to revolutionize areas such as healthcare and pharmaceutical. For example, a healthcare environment such as a large hospital or aged care could tag all pieces of medical equipment (e.g., scalpels, thermometers) and drug products for inventory management. Each

storage area or patient room would be equipped with RFID readers that could scan medical devices, drug products, and their associated cases. Such an RFID-based infrastructure could offer a hospital unprecedented near real-time ability to track and monitor objects and detect anomalies (e.g., misplaced objects) as they occur.

As personal health sensors become ubiquitous, they are expected to become interoperable. This means standardized sensors can wirelessly communicate their data to a device many people already carry today (e.g., mobile phones). It is argued by Lester et al. (2009) that one challenge in weight control is the difficulty of tracking food calories consumed and calories expended by activity. Then they present a system for automatic monitoring of calories consumed using a single body-worn accelerometer. To be fully benefited from such data for a large body of people, applying IoT technologies in such area would be a promising direction.

Mobile technology and sensors are creating ways to inexpensively and continuously monitor people's health. Doctors may call their clients to schedule an appointment, rather than vice versa, because the doctors could know their clients' health conditions in real time. Some projects for such purpose have been initiated. For example, EveryHeartBeat (<http://join.everyheartbeat.org/>) is a project for Body Computing to "connect the more than 5 billion mobile phones in the world to the health ecosystem." In the initial stage, heart rate monitoring is investigated. Consumers would be able to self-track their pulse, and studies show heart rate monitoring could be useful in detecting heart conditions and enabling early diagnosis. The future goal is to include data on blood sugar levels and other biometrics collected via mobile devices.

Energy

Home heating is a major factor in worldwide energy use. In IoT, home energy management applications could be built upon embedded Web servers (Priyantha et al. 2008). Through such online Web services, people can track and manage their home energy consumption. A system is designed by Gupta et al. (2009) for augmenting

these thermostats using just-in-time heating and cooling based on travel-to-home distance obtained from location-aware mobile phones. The system makes use of a GPS-enabled thermostat which could lead to savings of as much as 7%. In IoT, as things in homes would become smart and connected to the Internet, similar energy savings could be more effective. For example, by automatically sensing occupancy and sleep patterns in a home, it would be possible to save energy by automatically turning off the home's HVAC (heating, ventilation, and air conditioning) system.

Besides home heating, fuel consumption is also an important issue. GreenGPS, a navigation service that uses participatory sensing data to map fuel consumption on city streets, has been designed by Ganti et al. (2010). GreenGPS would allow drivers to find the most fuel-efficient routes for their vehicles between arbitrary end points. In IoT, fuel consumption would be further reduced by enabling cars and passengers to communicate with each other for ride sharing (Yuan et al. 2011).

Business

IoT technologies would be able to help to improve efficiency in business and bring other impacts on business (Mattern and Floerkemeier 2010):

- From a commercial point of view, IoT can help increase the efficiency of business processes and reduce costs in warehouse logistics and in service industries. This is because more complete and necessary information can be collected by interconnected things. Owing to its huge and profound impact on the society, IoT research and applications can also trigger new business models involving smart things and associated services.
- From a social and political point of view, IoT technologies can provide a general increase in the quality of life for the following reasons. Firstly, consumers and citizens will be able to obtain more comprehensive information. Secondly, care for aged and/or disabled people can be improved with smarter assistance

systems. Thirdly, safety can be increased. For example, road safety can be improved by receiving more complete and real-time traffic and road condition information.

- From a personal point of view, new services enabled by IoT technologies can make life more pleasant, entertaining, independent, and also safer. For example, business taking advantages of technologies of search of things in IoT can help locate lost things quickly, such as personal belongings, pets, or even other people (Tran et al. 2017).

Take improving information handover efficiency in a global supply chain as an example. The concept of *digital object memories* (DOM) is proposed in Stephan et al. (2010), which stores order-related data via smart labels on the item. Based on DOM, relevant life cycle information could be attached to the product. Considering the potential different stakeholders including manufacturer, distributor, retailer, and end customer along the supply/value chain, this approach facilitates information handover.

Further, there are many important bits of information in an IoT-based supply chain, such as the 5W (what, when, where, who, which). It is also necessary to integrate them efficiently and in real time in other operations. The EPCIS (Electronic Product Code Information System) network is a set of tools and standards for tracking and sharing RFID-tagged products in IoT. However, much of this data remains in closed networks and is hard to integrate (Wu et al. 2013). IoT technologies could be used to make it easier to use all this data, to integrate it into various applications, and to build more flexible, scalable, global application for better (even real-time) logistics.

Open Issues

The development of IoT technologies and applications is merely beginning. Many new challenges and issues have not been addressed, which require substantial efforts from both academia and industry. In this section, we identify some key

directions for future research and development from a data-centric perspective.

- *Data Quality and Uncertainty:* In IoT, as data volume increases, inconsistency and redundancy within data would become paramount issues. One of the central problems for data quality is *inconsistency detection*, and when data is distributed, the detection would be far more challenging (Fan et al. 2010). This is because inconsistency detection often requires shipping data from one site to another. Meanwhile, inherited from RFID data and sensor data, IoT data would be of great uncertainty, which also presents significant challenges.
- *Co-space Data:* In an IoT environment, the physical space and the virtual (data) space co-exist and interact simultaneously. Novel technologies must be developed to allow data to be processed and manipulated seamlessly between the real and digital spaces (Ooi et al. 2009). To synchronize data in both real and virtual worlds, large amount of data and information will flow between co-spaces, which pose new challenges. For example, it would be challenging to process heterogeneous data streams in order to model and simulate real world events in the virtual world. Besides, more intelligent processing is needed to identify and send interesting events in the co-space to objects in the physical world.
- *Transaction Handling:* When the data being updated is spread across hundreds or thousands of networked computers/smart things with differing update policies, it would be difficult to define what the transaction is. In addition, most of things are resource-constrained, which are typically connected to the Internet using lightweight, *stateless* protocols such as CoAP (Constrained Application Protocol) (<http://tools.ietf.org/html/draft-ietf-core-coap-18>) and 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) (<http://tools.ietf.org/wg/6lowpan>) and accessed using RESTful Web services. This makes transaction handling in IoT a great challenge. As pointed out by James et al. (2009) that the problem is that the world is

- changing fast, the data representing the world is on multiple networked computers/smart things, and existing database technologies cannot manage. Techniques developed for streamed and real-time data may provide some hints.
- *Frequently Updated Time-Stamped Structured (FUTS) Data:* The Internet, and hence IoT, contains potentially billions of Frequently Updated Time-Stamped Structured (FUTS) data sources, such as real-time traffic reports, air pollution detection, temperature monitoring, crops monitoring, etc. FUTS data sources contain states and updates of physical world things. Current technologies are not capable in dealing with FUTS data sources (James et al. 2009) because (i) no data management system can easily display FUTS past data, (ii) no efficient crawler or storage engine is able to collect and store FUTS data, and (iii) querying and delivering FUTS data is hardly supported. All these pose great challenges for the design of novel data management systems for FUTS data.
 - *Distributed and Mobile Data:* In IoT, data will be increasingly distributed and mobile. Different from traditional mobile data, distributed and mobile data in IoT would be much more highly distributed and data intensive. In the context of interconnecting huge numbers of mobile and smart objects, centralized data stores would not be a suitable tool to manage all the dynamics of mobile data produced in IoT. Thus there is a need for novel ways to manage distributed and mobile data efficiently and effectively in IoT.
 - *Semantic Enrichment and Semantic Event Processing:* The full potentials of IoT would heavily rely on the progress of Semantic Web. This is because things and machines should play a much more important role than humans in IoT to process and understand data. This calls for new research in semantic technologies. For example, there are increasing efforts in building public knowledge bases (such as DBpedia, FreeBase, Linked Open Data Cloud, etc.). But how these knowledge bases can be effectively used to

add to the understanding of raw data coming from sensor data streams and other types of data streams? To resolve this challenge, semantic enrichment of sensing data is a promising research direction. Further, consider the potential excessively large amount of subscriptions to IoT data. To produce proper semantic enrichment to meet different enrichment needs from different subscribers poses great challenges. Finally, how to effectively incorporate semantic enrichment techniques with semantic event processing to provide much better expressiveness in event processing is still at its initial stage. This will also demand a large amount of research efforts.

- *Mining:* Data mining aims to facilitate the exploration and analysis of large amounts of data, which can help to extract useful information for huge volume of IoT data. Data mining challenges may include extraction of temporal characteristics from sensor data streams, event detection from multiple data streams, data stream classification, activity discovery, and recognition from sensor data streams. Besides, clustering and table summarization in large data sets, mining large (data, information or social) networks, sampling, and information extraction from the Web are also great challenges in IoT.
- *Knowledge Discovery:* Knowledge discovery is the process of extracting useful knowledge from data. This is essential especially when connected things populate their data to the Web. The following issues related to knowledge discovery in IoT have been identified by (Weikum 2011; Tran et al. 2017): (i) automatic extraction of relational facts from natural language text and multimodal contexts; (ii) large-scale gathering of factual knowledge candidates and their reconciliation into comprehensive knowledge bases; (iii) reasoning on uncertain hypotheses, for knowledge discovery and semantic search; and (iv) deep and real-time question answering, e.g., to enable computers to win quiz game shows.
- *Security:* Due to the proliferation of embedded devices in IoT, effective device security

mechanisms are essential to the development of IoT technologies and applications (Huang et al. 2017). The National Intelligence Council (Anonymous 2008) argues that, to the extent that everyday objects become information security risks, the IoT could distribute those risks far more widely than the Internet has to date. For example, RFID security presents many challenges. Potential solutions should consider aspects from hardware and wireless protocol security to the management, regulation, and sharing of collected RFID data (Welbourne et al. 2009). Besides, it is argued by Lagesse et al. (2009) that there is still no generic framework for deploying and extending traditional security mechanisms over a variety of pervasive systems. Regarding security concerns of the network layer, it is suggested by Kounavis et al. (2010) that the Internet can be gradually encrypted and authenticated based on the observations that the recent advances in implementation of cryptographic algorithms have made general purpose processors capable of encrypting packets at high rates. But how to generalize such algorithms to IoT would be challenging as things in IoT normally only maintain low transmission rates and connections are usually intermittent.

- *Privacy:* Privacy protection is a serious challenge in IoT. One of the fundamental problems is the lack of a mechanism to help people expose appropriate amounts of their identity information. Embedded sensing is becoming more and more prevalent on personal devices such as mobile phones and multimedia cite layers. Since people are typically wearing and carrying devices capable of sensing, details such as activity, location, and environment could become available to other people. Hence, personal sensing can be used to detect their physical activities and bring about privacy concerns (Klasnja et al. 2009). Patient's electronic health record (EHR) is becoming medical big data with high volume Yang et al. (2018). The e-health IoT network is growing very fast, leading to big medical private data on the

Internet. There are many challenges, such as the information privacy, search, updating, and sharing, to be addressed.

- *Social Concerns:* Since IoT connects everyday objects to the Internet, social concerns would become a hot topic in the development of IoT. Further, online social networks with personal things information may incur social concerns as well, such as disclosures of personal activities and hobbies, etc. Appropriate economic and legal conditions and a social consensus on how the new technical opportunities in IoT should be used also represent a substantial task for the future (Mattern and Floerkemeier 2010). Trust management could play an important role in addressing social concerns (Lin and Dong 2018). With the help of trust models for social IoT, malicious behaviors can be detected effectively.

Summary

It is predicted that the next generation of the Internet will be comprised of trillions of connected computing nodes at a global scale. Through these nodes, everyday objects in the world can be identified, connected to the Internet, and take decisions independently. In this context, the Internet of Things (IoT) is considered as a new revolution of the Internet. In IoT, the possibility of seamlessly merging the real and the virtual worlds, through the massive deployment of embedded devices, opens up many new and exciting directions for both research and development. In this entry, we have provided an overview of some key research areas of IoT, specifically from a data-centric perspective. It also presents a number of fundamental issues to be resolved before we can fully realize the promise of IoT applications. Over the last few years, the Internet of Things has gained momentum and is becoming a rapidly expanding area of research and business. Many efforts from researchers, vendors, and governments have been devoted to creating and developing novel IoT applications. Along with the current research efforts, we encourage more insights into the problems of this promising

technology and more efforts in addressing the open research issues identified in this entry.

Cross-References

- ▶ [Big Data Analysis for Smart City Applications](#)
- ▶ [Big Data and Fog Computing](#)
- ▶ [Big Data in Smart Cities](#)
- ▶ [Big Data Stream Security Classification for IoT Applications](#)

References

- An J, Gui X, Zhang W, Jiang J, Yang J (2013) Research on social relations cognitive model of mobile nodes in internet of things. *J Netw Comput Appl* 36(2):799–810
- Anonymous (2008) National Intelligence Council (NIC), Disruptive civil technologies: six technologies with potential impacts on US interests out to 2025, Conference report CR 2008-07, Apr 2008. http://www.dni.gov/nic/NIC_home.html
- Ashton K (2009) That ‘Internet of Things’ thing. <http://www.rfidjournal.com/article/view/4986>
- Aztori L, Iera A, Morabito G (2010) The internet of things: a survey. *Comput Netw* 54(15):2787–2805
- CASAGRAS (2000) CASAGRAS (Coordination And Support Action for Global RFID-related Activities and Standardisation)
- European Commission (2009) European commission: internet of things, an action plan for Europe. http://europa.eu/legislation_summaries/information_society/internet/si0009_en.htm
- Fan W, Geerts F, Ma S, Müller H (2010) Detecting inconsistencies in distributed data. In: Proceedings of the 26th international conference on data engineering (ICDE), Long Beach. IEEE, pp 64–75
- Firouzi F, Rahmani AM, Mankodiya K, Badaroglu M, Merrett GV, Wong P, Farahani B (2018) Internet-of-things and big data for smarter healthcare: from device to architecture, applications and analytics. *Futur Gener Comput Syst* 78:583–586
- Ganti RK, Pham N, Ahmadi H, Nangia S, Abdelzaher TF (2010) GreenGPS: a participatory sensing fuel-efficient maps application. In: Proceedings of the 8th international conference on mobile systems, applications, and services (MobiSys), San Francisco. ACM, pp 151–164
- Guha S, Plarre K, Lissner D, Mitra S, Krishna B, Dutta P, Kumar S (2010) AutoWitness: locating and tracking stolen property while tolerating GPS and radio outages. In: Proceedings of the 8th international conference on embedded networked sensor systems (SenSys), Zurich. ACM, pp 29–42
- Guinard D (2010) A web of things for smarter cities. In: Technical talk, pp 1–8
- Guo B, Zhang D, Wang Z, Yu Z, Zhou X (2013) Opportunistic IoT: exploring the harmonious interaction between human and the internet of things. *J Netw Comput Appl* 36(6):1531–1539
- Gupta M, Intille SS, Larson K (2009) Adding GPS-control to traditional thermostats: an exploration of potential energy savings and design challenges. In: Proceedings of the 7th international conference on pervasive computing (pervasive), Nara. Springer, pp 95–114
- Huang Q, Yang Y, Wang L (2017) Secure data access control with ciphertext update and computation outsourcing in fog computing for internet of things. *IEEE Access* 5:12941–12950
- INFSO (2008) INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems. In: Co-operation with the working group RFID of the ETP EPOSS, internet of things in 2020, roadmap for the future, version 1.1, 27 May 2008
- ITU (2005) International Telecommunication Union (ITU) internet reports, The internet of things, Nov 2005
- James AE, Cooper J, Jeffery KG, Saake G (2009) Research directions in database architectures for the internet of things: a communication of the first international workshop on database architectures for the internet of things (DAIT 2009). In: Proceedings of the 26th British national conference on databases (BNCOD), Birmingham. Springer, pp 225–233
- Klasnja PV, Consolvo S, Choudhury T, Beckwith R, Hightower J (2009) Exploring privacy concerns about personal sensing. In: Proceedings of the 7th international conference on pervasive computing (pervasive), Nara. Springer, pp 176–183
- Kounavis ME, Kang X, Grewal K, Eszenyi M, Gueron S, Durham D (2010) Encrypting the internet. In: Proceedings of the ACM SIGCOMM conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), New Delhi. ACM, pp 135–146
- Lagesse B, Kumar M, Paluska JM, Wright M (2009) DTT: a distributed trust toolkit for pervasive systems. In: Proceedings of the 7th annual IEEE international conference on pervasive computing and communications (PerCom), Galveston. IEEE, pp 1–8
- Lai T-T, Chen Y-H, Huang P, Chu H-H (2010) PipeProbe: a mobile sensor droplet for mapping hidden pipeline. In: Proceedings of the 8th international conference on embedded networked sensor systems (SenSys), Zurich. ACM, pp 113–126
- Lester J, Hartung C, Pina L, Libby R, Borriello G, Duncan G (2009) Validated caloric expenditure estimation using a single body-worn sensor. In: Proceedings of the 11th international conference on ubiquitous computing (Ubicomp), Orlando. ACM, pp 225–234
- Li S, Xu LD, Zhao S (2014) The internet of things: a survey. *Information Systems Frontiers*, page (to appear)
- Lin Z, Dong L (2018) Clarifying trust in social internet of things. *IEEE Trans Knowl Data Eng* 30(2):234–248

- Mathur S, Jin T, Kasturirangan N, Chandrasekaran J, Xue W, Gruteser M, Trappe W (2010) ParkNet: drive-by sensing of road-side parking statistics. In: Proceedings of the 8th international conference on mobile systems, applications, and services (MobiSys), San Francisco. ACM, pp 123–136
- Mattern F, Floerkemeier C (2010) From the internet of computers to the internet of things. In: From active data management to event-based systems and more. Springer, Berlin, pp 242–259
- Ooi BC, Tan K-L, Tung AKH (2009) Sense the physical, walkthrough the virtual, manage the co (existing) spaces: a database perspective. SIGMOD Record 38(3):5–10
- Perera C, Zaslavsky AB, Christen P, Georgakopoulos D (2013) Context aware computing for the internet of things: a survey. CoRR, abs/1305.0982
- Priyantha NB, Kansal A, Goraczko M, Zhao F (2008) Tiny web services: design and implementation of interoperable and evolvable sensor networks. In: Proceedings of the 6th international conference on embedded networked sensor systems (SenSys), Raleigh. ACM, pp 253–266
- Ramanathan N, Schoellhammer T, Kohler E, Whitehouse K, Harmon T, Estrin D (2009) Suelo: human-assisted sensing for exploratory soil monitoring studies. In: Proceedings of the 7th international conference on embedded networked sensor systems (SenSys), Berkeley. ACM, pp 197–210
- Sheng QZ, Qin Y, Yao L, Benatallah B (2017) Managing the web of things: linking the real world to the web. Morgan Kaufmann, Amsterdam
- Stephan P, Meixner G, Koessling H, Floerchinger F, Ollinger L (2010) Product-mediated communication through digital object memories in heterogeneous value chains. In: Proceedings of the 8th annual IEEE international conference on pervasive computing and communications (PerCom), Mannheim. IEEE, pp 199–207
- Tran NK, Sheng QZ, Babar MA, Yao L (2017) Searching the web of things: state of the art, challenges and solutions. ACM Comput Surv (CSUR), 50(4):55:1–55:34
- Weikum G (2011) Database researchers: plumbers or thinkers? In: Proceedings of the 14th international conference on extending database technology (EDBT), Uppsala. ACM, pp 9–10
- Welbourne E, Battle L, Cole G, Gould K, Rector K, Raymer S, Balazinska M, Borriello G (2009) Building the internet of things using RFID: the RFID ecosystem experience. IEEE Internet Comput 13(3):48–55
- Wu Y, Sheng QZ, Shen H, Zeadally S (2013) Modeling object flows from distributed and federated RFID data streams for efficient tracking and tracing. IEEE Trans Parallel Distrib Syst 24(10):2036–2045
- Yan Z, Zhang P, Vasilakos AV (2014) Research on social relations cognitive model of mobile nodes in internet of things. J Netw Comput Appl 42:120–134
- Yang Y, Zheng X, Guo W, Liu X, Chang V (2018, in press) Privacy-preserving fusion of IoT and big data for e-health. Futur Gener Comput Syst
- Yuan J, Zheng Y, Zhang L, Xie X, Sun G (2011) Where to find my next passenger? In: Proceedings of the 13th international conference on ubiquitous computing (Ubicomp), Beijing. ACM, pp 109–118
- Zeng D, Guo S, Cheng Z (2011) The web of things: a survey (invited paper). J Commun 6(6):424–438
- Zimmerman T, Robson C (2011) Monitoring residential noise for prospective home owners and renters. In: Proceedings of the 9th international conference on pervasive computing (pervasive), San Francisco. Springer, pp 34–49

Big Data Analysis for Smart City Applications

Eugenio Cesario

Institute of High Performance Computing and Networks of the National Research Council of Italy (ICAR-CNR), Rende, Italy

Introduction

The twenty-first Century is frequently referenced as the “Century of the City”, reflecting the unprecedented global migration into urban areas that is happening nowadays (Butler 2010; Zheng et al. 2014). As a matter of fact, the world is rapidly urbanizing and undergoing the largest wave of urban growth in history. In fact, according to a United Nations report, urban population is expected to grow from 2.86 billion in 2000 to 4.98 billion in 2030 (UNR 2014): this translates to roughly 60% of the global population living in cities by 2030.

Such a steadily increasing urbanization has been leading to many big cities and bringing significant social, economic and environmental transformations. On the one hand it is modernizing people’s lives and giving increased opportunities offered in urban areas, on the other hand it is engendering big challenges in city management issues, such as large-scale resource planning, increased energy consumption, traffic congestion, air pollution, water quality, crime rising, etc. (Zheng et al. 2014).

Tackling these challenges seemed nearly impossible years ago given the complex and dynamic settings of cities. However, the widespread diffusion of sensing technologies and large-scale computing infrastructures has been enabling the collection of big and heterogeneous data (i.e., mobility, air quality, water/electricity consumption, etc.) that are daily produced in urban spaces (Al Nuaimi et al. 2015). Such big data imply rich knowledge about a city and are a valuable opportunity to achieve improvements in urban policies and management issues.

Motivated by the challenging idea of building more intelligent cities, several research projects are devoted to the development of Smart City applications. A Smart City is a city seeking to address public issues via ICT-based solutions on the basis of a multi-stakeholder, municipality based partnership (EUP 2017). The goal of building a smart city is to improve the quality of citizens' lives by using computer systems and technology to provide smart and efficient services and facilities. In this scenario, data analytics offers useful algorithms and tools for searching, summarizing, classifying and associating data turning it into useful knowledge for citizens and decision makers. Moreover, managing data coming from large and distributed sensing nodes poses significant technical challenges that can be solved through innovative solutions in parallel and distributed architectures, Cloud platforms, scalable databases, and advanced data mining algorithms and tools. Such issues pose several challenges and opportunities of innovations in urban environments, as described in the following sections of this article.

Research, Technological and Social Challenges in Smart Cities

A Smart City is a complex ecosystem that poses several research and technological challenges, which are complementary and connected among them. In fact, successful smart city deployments require a combination of methodological and technological solutions, ranging from the low-level sensors/actuators, effective data communi-

cations, data gathering, and analysis, finalized to effectively deliver smart applicative innovations to cities and citizens (Lea 2017). The most important data-related issues are briefly discussed in the following.

Networking and communications. The communication infrastructure of a smart city must support the connection among devices, buildings, facilities, people, and gather data and deliver services to myriad endpoints. Moreover, the heterogeneity of services in a smart city ecosystems requires a holistic approach to networking and communications, i.e. from low bandwidth wireless technologies to dedicated fiber optics as connection backbones. Such issues represent a technological challenge to deal with.

Cyber-physical systems and the IoT. Cyber-physical systems and the Internet of Things (IoT) are critical to the growth of smart cities, providing an urban connectivity architecture in which the objects of everyday life are equipped with micro-controllers, transceivers for digital communication, and suitable protocol stacks that will allow effective connectivity among devices and with users (Zanella et al. 2014). This enables the interaction among a wide variety of devices (i.e., home appliances, surveillance cameras, monitoring sensors) that collect enormous volume of data, making the smart city a digital data-rich ecosystem (Cicirelli et al. 2017).

Cloud computing. Cloud computing has a significant influence on the development of smart cities for two main reasons. First, it affects the way cities manage and deliver services, and enables a broader set of players to enter the smart city market. Second, the analysis of high-volume data (often to be processed in near real-time) takes crucial advantage from the scalable and elastic execution environments offered by Cloud architectures.

Open Data. Another significant challenge in smart cities is the adoption and exploitation of open data. Open data in the context of smart cities refer to public policy that requires or encourages public agencies to release data sets (e.g., citywide crime statistics, city service levels, infrastructure data) and make them freely accessible. The most important challenges associated with open data

include data security and in particular privacy issues. The evolution of open data represents a broadening of available information related to city operations, representing de-facto another source of city data.

Big data analytics. As above described, cities are collecting massive and heterogeneous amounts of data (text, video, audio), some of it is static but increasingly large parts are real-time. These data exhibit the classic characteristics of big data: volume, velocity (real-time generation), variety (extremely heterogeneous), veracity and value (very useful for business and research applications). For such a reason, a big effort is required to develop suitable data analytics methodologies to deal with such massive volumes of data, extremely heterogeneous in its sources, formats, and characteristics.

Citizen engagement. Citizen engagement represents a complementary aspect of smart cities and although not strictly a technical consideration, it concerns to make citizens feel into the “collective intelligence” of cities. A matter of fact, their willingness to exploit web applications (or other dialogue channels) for meaningful dialogue between cities and their citizens (i.e., reporting city issues, accidents, quality of services, etc.), is a crucial issue for the success of smart city initiatives.

Urban Big Data Collected in Smart Cities

Due to the complex and dynamic settings of cities, urban data are generated from many different sources in many different formats. A list of the most representative types of data and their brief description is reported in the following (Yuan et al. 2013; Zheng et al. 2014).

Traffic Data. Such a kind of data concerns the mobility of people and vehicles in urban and extra-urban areas. Traffic data can be collected in different ways, i.e. using loop sensors, surveillance cameras, and floating cars. Loop sensors are usually used to compute several mobility measures (i.e., speed of vehicles, traffic volume

on a road, etc.). Surveillance cameras are widely installed in urban areas, generating a huge volume of images and videos reflecting traffic patterns, thus providing a visual ground truth of traffic conditions to people. Floating car data are generated by vehicles traveling around a city with a GPS sensor, whose trajectories can be matched to a road network for deriving speeds on road segments. As many cities have already installed GPS sensors in taxicabs, buses, and logistics trucks for different purposes, floating car data have already been widely available and they have had higher flexibility and a lower deployment cost with respect to loop sensors and surveillance camera-based approaches.

Mobile Phone Data. Mobile telephone calls produce a huge amount of data every day. In fact, a call detail record contains attributes that are specific to a single instance of a phone call, such as the user’s location, start time, duration of the call, etc. The analysis of such kind of data allows users’ community detection, citywide human mobility discovery, etc.

Geographical Data. Geographical data describe the road network of the urban area under analysis. It is usually represented by a graph that is composed of a set of edges (denoting road segments) and a collection of geo-referenced nodes (standing for road intersections). Each edge can be enriched by several properties, such as the length, speed constraint, type of road, and number of lanes. Each node can also include information about point of interest (POI), i.e., restaurants, shopping malls, usually described by a name, address, category, and a set of geo-spatial coordinates; however, the information of POIs could vary in time (e.g., a restaurant may change its name, be moved to a new location, or even be shut down), so they need to be updated frequently. *Commuting Data.* Commuting data include data generated by people that leave an electronic footprint of their activities in urban environments. For example, people traveling in cities continuously generate digital traces of their passage, i.e., card swiping data in a subway/bus system, ticketing data in parking lots, records left by users when they borrow/return a bike in a bike sharing system, etc., thus producing huge

volume of commuting data that support decisions of urban planners and policy makers.

Environmental Data. Environmental data includes meteorological information (humidity, temperature, barometer pressure, wind speed, and weather conditions), air quality data (concentration of CO, CO₂, NO₂, PM_{2.5}, SO₂), environmental noise, etc. that can be collected by sensors placed in the city, portable sensors mounted on public vehicles, or air quality monitoring stations. There are also data collected by satellites that scan the surface of the earth with rays of different lengths to generate images representing the ecology and meteorology of a wide region.

Social Network Data. The huge volume of user-generated data in social media platforms, such as Facebook, Twitter and Instagram, are valuable information sources concerning human dynamics and behaviors in urban areas. In particular, social network data can be considered in two different forms. One is a social structure, represented by a graph, denoting the relationship, interdependency, or interaction among users. The other one is the user-generated social media data, such as texts, photos, and videos, which contain rich information about a user's behaviors/interests. Moreover, posts are often tagged with geo-graphical coordinates or other information (e.g., tags, photos) that allow identifying users' positions and build trajectories from geo-tagged social data. Therefore, social media users moving through a set of locations produce a huge amount of geo-referenced data that embed extensive knowledge about human mobility, crowded zones, interests and social dynamics in urban areas.

Economy Data. There is a variety of data representing a city's economic dynamics. For example, transaction records of credit cards, stock prices, housing prices, and people's incomes are valuable information to capture the economic rhythm of a city, therefore predicting which urban assets and features can positively (or negatively) influence future economy.

Energy Data. Energy data includes electricity consumption of buildings, energy costs, gas consumption of vehicles, etc. and provides funda-

mental information about city's resource usage. Such data can be obtained directly from sensors mounted on vehicles or in gas stations, by electricity and gas smart meters, or inferred from other data sources implicitly. Such data can be used to evaluate a city's energy infrastructures (e.g., the distribution of gas stations), find the most gas-efficient driving route, calculate the pollution emission from vehicles on road surfaces, optimize the energy electricity usage in residential areas, etc.

Health Care Data. A large volume of health care and disease data is generated by hospitals and clinics. In addition, the advances of wearable computing devices enable people to monitor their own health conditions, such as heart rate, pulse, and sleep time. Such data can be exploited for diagnosing a disease and doing a remote medical examination. In urban environments, this data can be aggregated to study the impact of environmental change on people's health, i.e. how urban noise impact people's mental health, how air pollution influences asthma-related diseases, etc.

Smart City Applications

A large number of smart city applications, exploiting urban data, have been implemented world-wide. A brief survey of the most important applications, grouped in seven categories, is reported below.

Smart Transportation. The joint analysis of traffic and commuting data provides useful knowledge to improve public and private mobility systems in a city. For example, intensive studies have been done to learn historical traffic patterns aimed at suggesting fast driving routes (Bejan et al. 2010; Herrera et al. 2010), at predicting real-time traffic flows (Herrera et al. 2010) and forecasting future traffic conditions on individual road segments (Castro-Neto et al. 2009). Mobility can also benefit in scenarios where traffic information collected through sensors, smart traffic lights and on-vehicle devices can be used to take action for specific critic events or to alleviate/resolve a traffic problem. Moreover, several applications are devoted to improve taxi services: some systems

provide taxi drivers with some locations (and routes) which they are more likely to pick up passengers quickly, while others schedule taxis to pick-up passengers via ride-sharing (subject to time, capacity and monetary constraints) maximizing the profit of the trip (Yuan et al. 2013; Ma et al. 2013). Finally, big data are also profitably used to build more reliable and efficient public transportation systems: for example, (i) real-time arrival time predictions of buses are more accurate by predicting traffic traces; (ii) bike sharing system operators may benefit from accurate models of bicycle flows in order to appropriately load-balance the stations throughout the day (Al Nuaimi et al. 2015; Zheng et al. 2014).

Smart Healthcare. Big data analysis is profitably exploited to provide smart healthcare solutions, which can improve the quality of patient care through enhanced patient handling. New systems can support a real-time monitoring and analysis of clinic data, or enable clinic parameters checking on demand or on a given frequency (i.e., hourly, daily or weekly). In particular, for certain patients' health issues, several benefits can be obtained from real-time data (blood pressure, cholesterol, sleeping patterns) monitoring through smart devices, which are connected to home or hospital for accurate and timely responses to health issues and for a comprehensive patient history records (Al Nuaimi et al. 2015; Zheng et al. 2014).

Smart Energy. The rapid progress of urbanization is consuming more and more energy, calling for technologies that can sense city-scale energy cost, improve energy infrastructures, and finally reduce energy consumption (Zheng et al. 2014). To this purpose, innovative systems are developed (i) to allow near-real forecasting of energetic demand through efficient analysis of big data collected, or (ii) to facilitate decision-making related to the supply levels of electricity in line with actual demand of the citizens and over all affecting conditions. Moreover, intelligent demand response mechanisms can also be profitably used to shift energy usage to periods of low demand or to periods of high availability of renewable energy. For example, intelligent algorithms (running at either device level or at community/trans-

former level) may enable a more efficient energy management to stay within community-assigned energy usage limits.

Smart Environment. The analysis of environmental data (i.e., urban air quality, noise, pollution) provides new knowledge about how environmental phenomena are influenced by multiple complex factors, such as meteorology, traffic volume, and land uses, or the impact that noise pollution has on people's mental and physical health, etc. (Zheng et al. 2014). Moreover, accurate weather forecasts can support country's agriculture effectiveness, it can inform people of possible hazardous conditions by supporting a more efficient management of energy utilization (Al Nuaimi et al. 2015). For example, a recent research study at MIT developed a real-time control system based on transport data and weather data to predict taxi demand with different weather conditions (Liv 2017).

Smart Safety and Security. Environmental disasters, pandemics, severe accidents, crime and terrorism attacks pose additional threats to public security and order. The wide availability of different kinds of urban data provides the opportunity to build predictive models with the ability, on one hand, to learn from history how to handle the aforementioned threats correctly and, on the other hand, to detect them in a timely manner or even predict them in advance (Zheng et al. 2014). For example, new technologies are enabling police departments to access growing volumes of crime-related data that can be analyzed to understand patterns and trends (Cesario et al. 2016), to anticipate criminal activity and thus to optimize public safety resource allocation (officers, patrol routes, etc.). Other studies are devoted to predict future environmental changes or natural disasters: some projects (Cesario and Talia 2010) are specifically aimed at developing new frameworks for earthquakes forecasting and to predict how the ground will shake as a result, which will give an opportunity to save lives and resources.

Smart Education. From assessment of graduates to online attitudes, each student generates a unique data track. By analyzing these data, education institutes (from elementary schools to universities) can realize whether they are using

their resources correctly and producing profitable didactic results. For example, data concerning student attendance, misconducts resulting in suspensions, college eligibility, dropout rate, on-track rate, etc., can be collected and analyzed to understand how different teaching strategies can address specific learning results. In addition, behavioural patterns can be extracted and exploited for student profiling, which will have positive effect on the knowledge levels and will suggest teaching/learning tools aimed at improving student's performance on the basis of his/her attitudes.

Smart Governance. Smart governance, whose term has been coined few years ago, is about using technological innovation to improve the performance of public administration, to enhance accountability and transparency, and to support politics in better planning and decision making. Specifically, political decisions can benefit from big data analytics models, which can support governments' focus on citizens' concerns related to health and social care, housing, education, policing, and other issues. In this way, more appropriate and effective decisions related to employment, production, and location strategies can be made. On the other hand, citizens can use governance data to make suggestions and enhance the quality of government services. Finally, information technology enables the integration and collaboration of different government agencies and combine or streamline their processes. This will result in more efficient operations, better handling of shared data, and stronger regulation management and enforcement (Al Nuaimi et al. 2015).

Future Directions for Research

Urban environments will continue generating larger and larger volumes of data, which will steadily represent a valuable opportunity for the development of Smart City applications and to achieve improvements in urban policies and management issues. This will lead towards several research directions aimed at making the storage, management and analysis of smart city

data more and more effective and efficient. From a methodological side, it is expected that more advanced algorithms for urban Big Data analysis will be studied through a higher integration with conventional city-related disciplines' theories; this will converge towards a multidisciplinary research field, where computer sciences meet traditional urban sciences (i.e, civil engineering, transportation, economics, energy engineering, environmental sciences, ecology, and sociology) to inspire data-driven smart city services. Furthermore, video and audio analytics methodologies, real-time data mining algorithms, advanced data-driven mobility applications, etc. will provide smart solutions to deal with the different conditions that dynamically change during the day in a city. From a technological side, advanced solutions for gathering, storing and analyzing data will be investigated: for example, further research in Cloud Computing, IoT, communication infrastructures will undoubtedly provide an essential support for innovative and scalable solutions able to capture, transmit, storage, search, and analyze data in so ever increasing data-rich cities.

Cross-References

- [Big Data Analysis for social good](#)
- [Big Data Analysis and IoT](#)

References

- Al Nuaimi E, Al Neyadi H, Mohamed N, Al-Jaroodi J (2015) Applications of big data to smart cities. *J Internet Serv Appl* 6(1):1–15
- Bejan AI, Gibbens RJ, Evans D, Beresford AR, Bacon J, Friday A (2010) Statistical modelling and analysis of sparse bus probe data in urban areas. In: 13th international IEEE conference on intelligent transportation systems, pp 1256–1263
- Butler D (2010) Cities: the century of the city. *Nature* 467:900–901
- Castro-Neto M, Jeong YS, Jeong MK, Han LD (2009) Online-svr for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert Syst Appl* 36(3, Part 2):6164–6173
- Cesario E, Talia D (2010) Using grids for exploiting the abundance of data in science. *Scalable Comput: Pract Exp* 11(3):251–262

- Cesario E, Catlett C, Talia D (2016) Forecasting crimes using autoregressive models. In: 2016 IEEE 2nd international conference on big data intelligence and computing, pp 795–802
- Cicirelli F, Guerrieri A, Spezzano G, Vinci A (2017) An edge-based platform for dynamic smart city applications. *Futur Gener Comput Syst* 76(C):106–118
- EUP (2017) World's population increasingly urban with more than half living in urban areas. Technical report, Urban Intergroup, European Parliament
- Herrera JC, Work DB, Herring R, Ban XJ, Jacobson Q, Bayen AM (2010) Evaluation of traffic data obtained via GPS-enabled mobile phones: the mobile century field experiment. *Trans Res C: Emerg Technol* 18(4):568–583
- Lea R (2017) Smart cities: an overview of the technology trends driving smart cities. Source: www.ieee.org
- Liv (2017) The live singapore! project. Source: senseable.mit.edu/livesingapore
- Ma S, Zheng Y, Wolfson O (2013) T-share: a large-scale dynamic taxi ridesharing service. In: 2013 IEEE 29th international conference on data engineering (ICDE), pp 410–421
- UNR (2014) World's population increasingly urban with more than half living in urban areas. Technical report, United Nations
- Yuan NJ, Zheng Y, Zhang L, Xie X (2013) T-finder: a recommender system for finding passengers and vacant taxis. *IEEE Trans Knowl Data Eng* 25(10):2390–2403
- Zanella A, Bui N, Castellani A, Vangelista L, Zorzi M (2014) Internet of things for smart cities. *IEEE Internet Things J* 1(1):22–32
- Zheng Y, Capra L, Wolfson O, Yang H (2014) Urban computing: concepts, methodologies, and applications. *ACM Trans Intell Syst Technol* 5(3):38:1–38:55

Big Data Analysis for Social Good

Robert M. Goerge
Chapin Hall at the University of Chicago,
Chicago, IL, USA

Overview

Big data is not just the 3, 4, or 5 Vs (Volume, Velocity, Variety, Veracity and Value) that characterize it but has become a catchall term for innovative uses of large datasets to gain additional benefit and build knowledge (Japec et al. 2015). How it is used can range from straightfor-

ward repackaging of it to make it user-friendly to using it to drive artificial intelligence applications. Much of what has been done with big data for social good revolves around making daily activities more information driven, such as mapping applications for daily commutes and providing better weather forecasts. These types of uses can benefit everyone in some way. This chapter focused on the analysis of big data to build knowledge that can benefit individuals and families at risk of harm, illness, disability, educational failure, or other special needs. This type of social good is focused on solutions to major ongoing social problems as opposed to the incremental improvement of daily living for the majority of the population (Coulton et al. 2015; Liebman 2018).

The Data

The primary source of big data to address social problems is the data from administrative information systems of government and nongovernmental agencies. Criminal justice, health care, income assistance, child and adult protective services, disability programs, employment, economic development, early childhood programs, K-12 schools, and postsecondary education are all example of programs that produce large datasets that can be used to produce evidence around who participates in these programs and how effective they might be. The data can range from data on who is enrolled in these programs to expenditures to detailed notes from the service provider on the characteristics of the individuals and the need they have. Many of the information systems for these programs have been in place since the 1980s, and they have become ubiquitous since the turn of the century. Therefore, data on hundreds of millions of individuals with billions of transactions is now available.

However, unlike increasingly more examples of data that is open and accessible on government websites, the type of data that is needed to conduct rigorous research and analysis is not widely accessible outside of the organizations

that collect them (Goerge 2018). The value to these data is that they are person-specific and provide information about health conditions, negative behaviors, educational attainment, employment and income, welfare program receipt, and violent behavior. Therefore, they are quite sensitive and could be used to harm people. At a minimum, they can be used to stigmatize individuals. Therefore, these data are kept as secure, and access to them is only provided when there is a clear benefit over the potential risks.

These are the data that are a potential boom for building evidence about the effectiveness of public policies and programs (Card et al. 2010). Social scientists are anxious to find ways to access these data so that more evidence can be developed to improve the social good. The Commission on Evidence-Based Policy, a bipartisan effort of the US Congress, signed into being by President Obama, provides one blueprint for increasing evidence and maintaining confidentiality of sensitive data (Commission on Evidence-Based Policymaking 2017).

As the tools that can increase secure access have become more available, access to these data has increased. In addition, best practices have been established to put forth protections against unlawful disclosure when data is shared with an organization or individual external to the data collection agencies.

Combining Data

Each individual dataset provides one piece of an individual's experience in a system of services or in a school. By combining datasets, one can enhance the utility of the data. For example, combining employment and education data can provide insight into one kind of educational trajectories lead to higher incomes. However, in the United States, there is no one identifier that is used by every government agency to identify an individual. While the Social Security Number has become a default national identification number, many states prohibit the use of the SSN for identification purposes.

Computational Power

As computational power has increased, a whole new set of analytic techniques and new algorithms are possible (Wang et al. 2016). This is because computers can now process a greater amount of data through many more iterations allowing for models to be developed that in the past would not be possible. Therefore, larger datasets, built through combining data from multiple sources, can be built that describe more characteristics of more individuals and the contexts in which they live. Analyses of large datasets that could have taken weeks to run just a decade ago can now be ran in minutes. Therefore, the ability to take advantage of these large datasets is no longer limited by technology but only non-technological issues.

An example of this is an analysis and replication completed by the author and his colleagues. Records on 3 million individuals in the Supplemental Nutritional Assistance Program (SNAP) and child protective services (CPS) system in Illinois were combined into families based on relationship information contained in the state information systems. Data on family member interactions with the criminal justice and juvenile justice system, child protective services (foster care), mental health services, and substance abuse treatment were connected to these individuals. We learned that 23 percent of the families accounted for 86 percent of the expenditures across these 5 program areas. This work was originally done for individuals in SNAP and CPS in 2008. It took roughly 1 week to create the analytic file, using SAS. When we replicated this work in 2016, because of programming the algorithm in Python, the dataset creation step took a matter of hours.

Similarly, cloud computing and the ability to ensure security is being embraced by the federal government and increasingly by state governments (Foster 2018). This is potentially a game changer since datasets can be stored in the cloud and data providers can provide permission for either internal or external analysts to access those data with confidence that there are sufficient controls for monitoring access. One such initia-

tive is New York University and University of Chicago's Administrative Research Data Facility, also known as the Coleridge Initiative (Commission on Evidence-Based Policymaking 2017) (<https://coleridgeinitiative.org/>).

Non-technological Barriers

Given the sensitivity of these data, not only to those individuals in the data but to the organizational leaders who could potentially be negatively affected in multiple ways by disclosure of the data, there are rules and regulations specific to each dataset that govern the uses of the data (Petrla 2018). For the most part, the use of the data should promote the administration of the program. For example, the Family Educational Rights and Privacy Act (FERPA) allows disclosure of personally identifiable information if the purpose is to "improve instruction." Determining whether or not the use has the potential to improve instruction is not clearly established and open to interpretation by each educational entity. Therefore, access to these data is predominantly still relied on developing relationships with the data providers (Goerge 2018).

Given the ambiguity of many of the rules and regulations around the use of sensitive administrative data, beyond keeping the data secure, it is necessary to partner with data providing agencies in a way that can both build trust and also ensure that the data providing agency can benefit from any analysis done. Trust involves multiple factors that must be addressed over time, but a process whereby the data providers can review the work of analysts is critical to ensure not only the proper use of the data but also that data providers are simply aware of where analysis of their information is disseminated.

Analysis of These Data

The types of analyses that are done with these data to support the social good are expanding at a significant rate. Predictive analytics are beginning to be used to affect frontline decision-

making. For example, Allegheny County in Pennsylvania is using predictive risk models to determine if a child is going to be re-referred to child protective services. This information can be used to determine if services should be provided at the time of a call to child protective services (Hurley 2018). It is safe to say that there are high stakes around these decisions and the receptivity of frontline workers to use these data, when they are shouldering the risks of these decisions, may be less than enthusiastic. Careful evaluations and the need to ensure that no harm comes as a result of decisions driven by big data are of the utmost importance.

Coulton et al. (2015) provides a number of examples of data analysis that have been conducted in welfare programs, child welfare services, homelessness programs, Medicaid, and participation in multiple programs. Multiple traditional designs, such as randomized controlled trials and quasi-experimental studies, are also being used. While it is clear that these efforts will provide significant additional knowledge, translating these findings to action – to promote social good – is not a trivial task. "Translational data science" is the term that speaks to how the data science can improve services and achieve social good and improvements in social welfare (<https://cdis.uchicago.edu/news/2017/4/21/translational-data-science-workshop>).

With increased use of machine learning techniques, such as decision trees, dimension reduction methods, nearest neighbor algorithms, support vector models, and penalized regression, with administrative data, new possibilities for impact on social good are on the horizon (Hindman 2015). Research organizations and universities are preparing to apply these methods to improve the social good. One caution, which many have expressed, is that it is important to understand the interventions and processes that professionals, from medical professionals to social works to teachers, use to address specific conditions. Simply processing big data can lead to spurious correlations on one hand and recommendations that are completely unfeasible on another. Many professionals are very constrained in what they can do with their data, even if they know that

there are confirmed relationships between particular characteristics of individuals and future outcomes. We live in a country where confidentiality and privacy are predominant social values and where research ethics must be practiced (Commission on Evidence-Based Policymaking 2017).

A lack of understanding of how a particular analysis or algorithm may be applied could lead to unintended consequences (Sandvig et al. 2016). The issue of race is one of the most sensitive. Often, because of how much variance, without rich data, a race variable can explain; too much is attributed to this characteristic. It can lead to profiling and inappropriate targeting and, at a minimum, ineffective practices.

Conclusion

Whether the innovative analyses of big data have caught up to the ethical use of those analyses is an open question. As data scientists attempt to implement innovations based on analytics in high stakes domains, such as health and human services, great caution must be taken to ensure that no harm is done. While this is not new to be concerned about potential harms, the algorithms used may not be sufficiently transparent to make clear what potential actions might cause harm or action based on race, gender, sexuality, ethnicity, disability, or some other characteristics.

References

- Card D, Chetty R, Feldstein M, Saez E (2010) Expanding access to administrative data for research in the United States. NSF SBE 2020 White Paper
- Coulton CJ, Goerge R, Putnam-Hornstein E, de Haan B (2015). Harnessing big data for social good: A grand challenge for social work. American Academy of Social Work and Social Welfare, Paper No. 11.
- Foster I (2018) Research infrastructure for the safe analysis of sensitive data. Ann Am Acad Pol Soc Sci 675(1):102–120. <https://doi.org/10.1177/0002716217742610>
- Goerge RM (2018) Barriers to accessing state data and approaches to addressing them. Ann Am Acad Pol Soc Sci 675(1). <https://doi.org/10.1177/0002716217741257>

Hindman M (2015) Building better models: prediction, replication, and machine learning in the social sciences. Ann Am Acad Pol Soc Sci 659(May):48–62. <https://doi.org/10.1177/0002716215570279>

Hurley D (2018) Can an algorithm tell when kids are in danger? Times, New York

Japec L, Kreuter F, Berg M, Biemer P, Decker P, Lampe C (2015) AAPOR report on big data

Liebman JB (2018) Using data to more rapidly address difficult U.S. social problems. Ann Am Acad Pol Soc Sci 675(1):166–181. <https://doi.org/10.1177/0002716217745812>

Petrila J (2018) Turning the law into a tool rather than a barrier to the use of administrative data for evidence-based policy. Ann Am Acad Pol Soc Sci 675(1):67–82. <https://doi.org/10.1177/0002716217741088>

Policy-Making, Commission on Evidence-Based Policymaking. 2017. The promise of evidence-based policymaking. Washington, DC: Commission on Evidence-Based Policymaking. Available from <https://www.cep.gov/content/dam/cep/report/cep-final-report.pdf> (accessed 14 October 2017).

Sandvig C, Hamilton K, Karahalios K, Langbort C (2016) Automation, algorithms, and politics | when the algorithm itself is a racist: diagnosing ethical harm in the basic components of software. Int J Commun 10(0):19

Wang C, Cen M-H, Schifano E, Jing W, Yan J (2016) Statistical methods and computing for Big Data. Stat Interface 9(4):399–414. <https://doi.org/10.4310/SII.2016.v9.n4.a1>

Big Data Analysis in Bioinformatics

Mario Cannataro

Data Analytics Research Center, University “Magna Græcia” of Catanzaro, Catanzaro, Italy

Synonyms

Bioinformatics; Bioinformatics algorithms; Bioinformatics big data

Definitions

Genomics “Genomics describes the study of all of a person’s genes (the genome), including interactions of those genes with each other

and with the person's environment" (Source National Human Genome Research Institute).

Proteomics "Proteomics describes the study of all the proteins (the proteome) in an organism, tissue type, or cell" (Source National Human Genome Research Institute).

Interactomics "Interactomics describes the study the whole set of molecular interactions (the interactome) in an organism, tissue type, or cell."

Pharmacogenetics "Pharmacogenetics is the field of study dealing with the variability of responses to medications due to variation in single genes" (Source National Human Genome Research Institute).

Pharmacogenomics "Pharmacogenomics is similar to pharmacogenetics, except that it typically involves the search for variations in multiple genes that are associated with variability in drug response" (Source National Human Genome Research Institute).

Overview

The chapter discusses big data aspects of bioinformatics data with special focus on omics data and related data analysis pipelines. Main challenges related to omics data storage, preprocessing and analysis are presented. After describing main omics data, such as genomics, proteomics and interactomics data, the chapter describes main biological databases, and presents several data analysis pipelines for analyzing big omics data that attempt to face big data challenges.

Introduction

Omics disciplines such as genomics, proteomics, and interactomics, to cite a few, are at the core of modern biomedical research. Omics studies are enabled by three main factors: (i) the availability of high-throughput experimental platforms, such as microarray, mass spectrometry, and next generation sequencing, that produce an overwhelming amount of experimental data; (ii) the integration of omics data with clinical data (e.g., bioimages, pharmacology data, etc.); (iii) the development of bioinformatics algorithms and comprehensive data analysis pipelines that preprocess, analyze, and extract knowledge from such data. The peculiar characteristics of big data in the context of bioinformatics can be declined as follows.

- **Volume:** the volume of big data in bioinformatics is having an explosion, especially in healthcare and medicine, due to many reasons: high resolution of experimental platforms that may produce terabytes of omics data, increasing number of subjects enrolled in omics studies, increasing digitalization of healthcare data (laboratory tests, visits to the doctor, administrative data about payments) that in the past was stored on paper.
- **Velocity:** in bioinformatics new data is created at increasing speed due to technological advances in experimental platforms and due to increased sampling rate of sensors for health monitoring. This requires that data are ingested and analyzed in near real time. As an example, considering the medical devices used to monitor patients or the personal sensors used by general subjects to collect data, big data analytics platforms need to analyze that data and to transmit results to medical doctors or to the user itself in near real time. Moreover, the use of the IoT (Internet of Things) devices in medicine will increase velocity of big data in bioinformatics.
- **Variety:** there is a great variety and heterogeneity of data in bioinformatics coming from medicine, biology, and healthcare. Other than raw experimental omics data, there are clinical data, administration data, sensors data, and social data that contribute to variety.
- **Variability:** how the data is captured in bioinformatics may vary due to several factors, such as experiment setup, use of standard for-

mats or proprietary formats, and use of local conventions in the collection of data. Thus variability may lead to wrong interpretation of data.

- **Value:** the value of big data produced in bioinformatics is related not only to the cost of infrastructures to produce (experimental platforms) and to analyze (software tools, computing infrastructures) them but also to the biomedical knowledge it is possible to extract from data.

The emerging of this big data trend in bioinformatics poses new challenges for computer science solutions, regarding the efficient storage, preprocessing, integration, and analysis of omics and clinical data that is today the main bottleneck of the analysis pipeline. To face those challenges, main trends are (i) use of high-performance computing in all steps of analysis pipeline, including parallel processing of raw experimental data, parallel analysis of data, and efficient data visualization; (ii) deployment of data analysis pipelines and main biological databases on the Cloud; (iii) use of novel data models that combine structured (e.g., relational data) and unstructured (e.g., text, multimedia, biosignals, bioimages) data, with special focus on graph databases; (iv) development of novel data analytics methods such as sentiment analysis, affective computing, and graph analytics that overcome the limits of traditional statistical and data mining analysis; (v) particular attention to issues regarding privacy of data and users, as well as permitted ways to use and analyze biomedical data (e.g., with extensive development of anonymization techniques and privacy-preserving rules and models). The rest of the chapter first introduces big omics data in bioinformatics, then describes main biological databases, and finally presents several data analysis pipelines for analyzing omics data.

Big Data in Bioinformatics

This section introduces the most relevant big data generated in bioinformatics. Such data are mainly

generated through high-throughput experimental platforms (e.g., microarray, mass spectrometry, next generation sequencing) that produce the so-called *omics data* (e.g., genomics, proteomics, and interactomics data). The volume of these data is increasingly huge because the resolution of such platforms is increasingly high and because the biomedical studies usually involve thousands of biological samples.

Microarray Data

Microarrays allow to detect, among the others, gene expression data, single nucleotide polymorphism (SNP) data, and drug metabolism enzymes and transporters (DMET) SNP data. A main difference among such data is that gene expressions are represented as numerical data, while SNPs are encoded as strings. The analysis of microarray data presents several challenges (Guzzi and Cannataro 2011) that are outlined in the following.

Gene Expression Data

Main biological processes are carried out by genes and proteins. According to the central dogma of molecular biology, genes encode the formation of proteins, and, in particular, genes regulate the formation of proteins through the transcription process which produces RNA (ribonucleic acid). The expression level of RNA (or mRNA – messenger RNA) is an indirect measure about the production of the proteins. Microarrays allow to study the expression level of the RNA associated to several genes (and thus their potential protein transcriptions) in a single experiment. Moreover, microarrays also allow the study of the microRNA (miRNA) that are short noncoding molecules of RNA that contain about 22 nucleotides and have a role in various functions and especially in regulation of gene expression (Di Martino et al. 2015). In a typical microarray experiment, first the RNA is extracted from a biological sample, and then it is put onto the microarray chip that contains a set of probes to bind the RNA. Probes are formed by oligonucleotides or complementary DNA (c-DNA). A light source is used to bleach the fluorescent markers, and then the results are stored in a high-definition image. At this

point, a preprocessing phase is needed to remove the noise, recognize the position of different probes, and identify corresponding genes. The correspondence among probes and genes is usually encoded into a location file that needs to be used to correctly locate the expression level of each gene. Starting from a raw microarray data (named .CEL file in the Affymetrix terminology), the resulting preprocessed file contains the expression level of each gene (among the analyzed genes) in a sample. Joining all the data related to several samples, it is possible to obtain a matrix where each column indicates a sample, each row indicates a gene, and the element (i, j) contains the expression level of gene i in sample j .

SNP Data

DNA is made up of four bases called adenine (A), cytosine (C), guanine (G), and thymine (T). These elements are organized as double helices that form long chains called chromosomes. Each chromosome contains a DNA molecule, and each DNA molecule is made up of many genes that are segments of DNA. Genes are then translated into proteins by the mediation of the ribonucleic acid (RNA) that has a simpler structure with respect to the DNA and is made of a single filament made up of four subunits, called adenine (A), cytosine (C), guanine (G), and uracil (U). Each individual has a unique sequence of DNA that determines his/her characteristics, and differences among sequences are measured in terms of substitutions of bases in the same position. The substitution of a single base that occurs in a small subset of population is named mutation or single nucleotide polymorphism (SNP), e.g., if we consider the sequence (ATGT) and the modified one (ACGT), then the SNP is denoted with the symbol T/C. A SNP dataset is thus constituted by a table where each row indicates a gene or a specific portion of a gene, each column indicates a sample (patient), and a cell (i, j) contains the SNP, if any, on the gene i , detected in sample j . SNPs data are represented by strings over the A, C, G, T alphabet. A SNP marker is just a single base change in a DNA sequence, and several works demonstrated the correlation

between SNPs and diseases or between the presence of SNPs and response to drugs (Wollstein et al. 2007).

DMET Data

Pharmacogenomics is a subfield of genomics that studies how variations in genes (e.g., SNPs) are responsible for variation in the metabolism of drugs, and it also investigates the so-called adverse drug reactions (ADR) that happen when a drug has a narrow therapeutic index (a measure of the amount of drug that may cause lethal effect), i.e., the difference between the lethal and the therapeutic dose of the drug is little. The Affymetrix DMET (drug metabolism enzymes and transporters) platform (Arbitrio et al. 2016b) analyzes simultaneously the polymorphisms of 225 ADME-related genes, i.e., genes known to be related to drug absorption, distribution, metabolism, and excretion (ADME) (Li et al. 2010). In particular, the current Affymetrix DMET microarray chip allows the investigation of 1936 probes, corresponding to 1936 different nucleotides each one representing a portion of the genes having a role in drug metabolism. A DMET SNP dataset is a table where each row indicates a gene or a specific portion of a gene, each column indicates a sample (patient), and a cell (i, j) contains the SNP, if any, on the ADME gene i , detected in sample j .

The importance of the analysis of DMET data is demonstrated by different works that evidenced the correlation between genetic variations in ADME genes and different effects of drug treatments (Di Martino et al. 2011; Arbitrio et al. 2016a).

Mass Spectrometry Data

Proteomics is about the study of the proteins expressed in an organism or a cell. Computational proteomics is about the algorithms, databases, and methods used to manage, store, and analyze proteomics data (Cannataro 2008). Mass spectrometry (MS), the main experimental platform for proteomics, uses the mass spectrometer to investigate the proteins expressed in a sample (Aebersold and Mann 2003). In a very intuitive way, MS measures the mass (in Dalton) of all

molecules contained in a sample. More precisely, the output of a MS experiment, said spectrum, is a long sequence of pairs of numbers (i, m_z), where m_z is the mass-to-charge ratio of a molecule and i is the intensity or abundance of that molecule. The manual inspection of spectra is unfeasible, so several algorithms to manage and analyze them are available. An important role is played by preprocessing algorithms that regard the way how spectra are cleaned, preprocessed, organized, and stored in an efficient way. Computational proteomics includes (i) qualitative proteomics, i.e., the identification of the sequence of the detected proteins; (ii) quantitative proteomics, i.e., the determination of the relative abundance of proteins in a sample; and (iii) biomarker discovery, i.e., finding most discriminant features among spectra coming from two different populations (e.g., healthy versus diseased).

Interaction Data

Interactomics is a recent omics discipline that refers to the study of the *interactome*, i.e., the list of all the interactions in an organism (Cannataro et al. 2010b). Interactomics focuses on the modeling, storage, and retrieval of protein-to-protein interactions (PPI) and protein interaction networks (PINs). Interactomics is important to explain and interpret protein functions because many of them are performed when proteins interact with each other. If interactions are stable along the time, they form the so-called *protein complexes*, and computational methods of interactomics may be used to discover and predict protein complexes. Wet lab experiments allow to find binary interactions that involve only two proteins, or multiple interactions, such as protein complexes. Computational (in silico) methods of interactomics may predict novel interactions without the execution of wet lab experiments. PPIs are often stored in databases as a list of binary interactions in the form (P_i, P_j). Such databases offer efficient storing and retrieval of single PPI data or the entire PIN, using expressive querying mechanisms. A way to represent all PPIs of an organism is through a graph named protein interaction network (PIN) (Cannataro and Guzzi 2012). The nodes of a PIN are the proteins, while the edges

represent the interactions among them. The PINs are represented through undirected graphs, but sophisticated models use directed and labeled edges that carry on additional information on each interaction. The PIN of an organism is built by extracting all the binary interactions (P_i, P_j) from a PPI database and by building the related graph. A key activity of interactomics is the study of the PIN of an organism or the comparison of the PINs of different organisms. In summary, computational methods of interactomics regard the prediction, storage, and querying of PPIs data and the analysis and comparison of PINs graphs.

Biological Databases

This section presents main databases containing information about biological data, with special focus on protein sequences, structures, and interactions.

Sequences Databases

Sequences databases contain the primary structure of proteins (i.e., the amino acids sequence) and related annotations, such as the experiment and scientist that discovered them. They offer a querying interface that can be interrogated using a protein identifier or a sequence fragment, e.g., to retrieve similar proteins. Similar proteins to a target protein are retrieved by using a matching algorithm named *sequence alignment*.

The EMBL Nucleotide Sequence Database

The EMBL (European Molecular Biology Laboratory) Nucleotide Sequence Database (Boeckmann et al. 2003) stores primary nucleotide sequences, and, through an international collaboration, it is synchronized with the DDBJ (DNA Data Bank of Japan) and GenBank databases. The database contains information about the CDS (coding DNA sequence), i.e., the coding region of genes and other information. Currently it contains more than three billions nucleotides and more than 1.3 millions sequences. Data are made available as txt, XML, or FASTA. FASTA is a standard format

to represent nucleotide sequences or peptide sequences, where nucleotides or amino acids are represented with single-letter code. As an example, the FASTA format of TC5b (Trp-Cage Miniprotein Construct, accession number 1L2Y) protein, one of the smallest protein with only 20 amino acids, is the following, where the first row contains metadata and the second one contains the sequence:

```
>1L2Y:A|PDBID|CHAIN|SEQUENCE\\
NLYIQWLKDGGPSSGRPPPS
```

Uniprot Database

Swiss-Prot (Boeckmann et al. 2003) was one of the first database storing protein sequences, and it was characterized by the use of flat files, the manual curation of data to avoid redundancies, and an extensive use of annotations. In 2002 Swiss-Prot was merged with the Tr-EMBL (Boeckmann et al. 2003) and the PSD-PIR (Barker et al. 1999) databases, resulting in the UniProt (Universal Protein Resource) database (Consortium 2010), that contains a large volume of protein sequences and functional information. UniProt contains (i) the UniProt Knowledgebase (UniProtKB) that is the access point for information about protein function, classification, and cross-references; (ii) the UniProt Reference Clusters (UniRef); and (iii) the UniProt Archive (UniParc). In particular, UniProtKB comprises the UniProtKB/Swiss-Prot section that is manually annotated and reviewed and the UniProtKB/Tr-EMBL section that is not reviewed and is automatically annotated. UniProt database is publicly accessible at www.uniprot.org.

dbSNP

The Single Nucleotide Polymorphism database (dbSNP) (Sherry et al. 2001) contains several molecular variations such as single nucleotide polymorphisms (SNPs), short deletion and insertion polymorphisms (indels/DIPs), microsatellite markers or short-tandem repeats (STRs), multi-nucleotide polymorphisms (MNPs), heterozygous sequences, and named variants. Currently it contains several millions distinct variants for many organisms, including *Homo sapiens* and other

species. dbSNP is publicly accessible at www.ncbi.nlm.nih.gov/SNP/.

PharmGKB

The Pharmacogenomics Knowledge Base (PharmGKB) (Hewett et al. 2002) stores data collected from several pharmacogenetic works, including genomic, phenotype, and clinical information. Users can browse, query, and download the knowledge base, and they can extract pathways and clinical information. PharmGKB stores pathways related to drugs that have pharmacogenetic role, including pharmacokinetic pathways (what the body does to the drug) or pharmacodynamic pathways (what the drug does to the body). Cross-links are connected to pharmacogenomic concepts forming the knowledge base for pharmacogenomic researchers. PharmGKB is publicly accessible at www.pharmgkb.org.

Structures Databases

Structures databases contain the secondary and tertiary structure of proteins and several tools for 3D visualization of proteins. The secondary structure of proteins refers to the way in which the amino acids are organized into well-defined structures such as helix, sheets, or random coil. It is represented as a string of characters belonging to the Dictionary of Protein Secondary Structure (DSSP) having the same length of the primary structure (i.e., the amino acids sequence). DSSP codes eight types of secondary structures, e.g., H (4 – *turnhelix* or *α helix*), E (*β sheet*), C (*coil*), and so on. The tertiary structure of proteins refers to the way in which the amino acids (each atom) are organized into the 3D space. It contains the *x*, *y*, *z* coordinates of each atom of all amino acids. Distances are expressed in Angstroms. Relevant annotations regard the methods used to discover the 3D structure, e.g., X-ray crystallography or nuclear magnetic resonance (NMR). Similar proteins to a target protein are retrieved by using a matching algorithm named *structure alignment*.

Protein Data Bank (PDB)

The Protein Data Bank (PDB) (Berman et al. 2000) contains structural data of biological

macromolecules generated by using crystallography and nuclear magnetic resonance (NMR) experiments. Each PDB entry is stored in a single flat file. PDB is publicly accessible at www.rcsb.org. As an example, the sequence and secondary structure for protein 1L2Y (chain A) is reported below (empty, means no structure assigned; *H*, α -helix; *G*, 3/10helix).

NLYIQWLKD GPSSGRPPPS
HHHHHHHTT GGGGT

Proteomics Databases

Proteomics databases considered here are those containing mass spectrometry data.

Global Proteome Machine Database

The Global Proteome Machine Database (Craig et al. 2004) is a system for storing, analyzing, and validating proteomics tandem mass spectrometry (MS/MS) data. The system implements a relational database and uses a model based on the Hypo-PSI Minimum Information About Proteomic Experiment (MIAPE) (Taylor et al. 2006) scheme. The Global Proteome Machine Database is publicly accessible at www.thegpm.org.

Peptide Atlas

PeptideAtlas (Desiere et al. 2006) is a database that stores mass spectrometry experiments and data about peptides and proteins identified by liquid chromatography tandem mass spectrometry (LC-MS/MS). PeptideAtlas uses spectra as primary information to annotate the genome; thus, the operation of this database involves these main phases: (i) identification of peptides through LC-MS/MS, (ii) genome annotation using peptide data, and (iii) storage of such genomics and proteomics data in the PeptideAtlas database. Users can query the system using a web interface or can download the database. The Global Proteome Machine Database is publicly accessible at www.peptideatlas.org.

PRIDE

The PRoteomics IDEntifications database (PRIDE) stores protein and peptide identifica-

tions and all information about the performed mass spectrometry experiment (Jones et al. 2006). Protein identifications are identified by unique accession numbers and accompanied by a list of one or more peptide identifications and related sequence and position of the peptide within the protein. Peptide identifications are encoded using the PSI (Proteomic Standard Initiative) mzData format and its evolutions. PRIDE is publicly accessible at www.ebi.ac.uk/pride.

Protein-Protein Interactions Databases

Databases of experimentally determined interactions obtained through high-throughput experiments, and databases of predicted interactions, obtained through in silico prediction, are presented in the following.

DIP

The Database of Interacting Proteins (DIP) contains interactions experimentally determined in different organisms, including Escherichia coli, Rattus norvegicus, Homo sapiens, Saccharomyces cerevisiae, Mus musculus, Drosophila melanogaster, and Helicobacter pylori. The DIP database is implemented as a relational database, and it stores data about proteins, experiments, and interactions data. The entry of new interactions is manually curated, and interactions need to be described in peer-reviewed journals. Users have different ways to query DIP through a web-based interface: (i) *Node*, by specifying a protein identifier; (ii) *BLAST*, by inserting a protein sequence that is used by DIP to retrieve all the matching proteins; (iii) *Motif*, by specifying a sequence motif described as regular expression, (iv) *Article*, by inserting an article identifier that is used by DIP to search for interactions described in that article; and (v) *pathBLAST*, by inserting a list of proteins composing a pathway that are used by DIP to extract all the interaction pathways that align with the query pathway. DIP also allows to export the whole PIN of an organism using different formats, such as PSI-MI XML and PSI-MI TAB. DIP is publicly accessible at dip.mbi.ucla.edu/dip/.

BIND

The Biomolecular Interaction Network Database (BIND) (Alfarano et al. 2005) contains experimentally determined protein interactions annotated with information about molecular function extracted from literature. Supported organisms include *Homo sapiens*, *Saccharomyces cerevisiae*, *Mus musculus*, and *Helicobacter pylori*. BIND contains three main data types: interaction, molecular complex, and pathway. An interaction record stores the reaction event between two objects, e.g., proteins. A molecular complex record stores the interactions, temporally sorted, producing it. A pathway, defined as a network of interactions that mediate some cellular functions, is described as a series of reactions. The database supports different query methods: by inserting identifiers from other biological databases or by providing search terms about literature, structure, and gene. Results are visualized with the BIND Interaction viewer.

I2D

The Interologous Interaction Database (I2D) (Brown and Jurisica 2007) contains predicted interactions between human proteins. It combines interaction data derived from literature, from a set of curated journals, and from databases such as MINT or BIND, using predictions from other organisms (e.g., *Saccharomyces cerevisiae*, *Caenorhabditis elegans*, *Drosophila melanogaster*, and *Mus musculus*). The prediction algorithm inside I2D is based on the hypothesis that patterns of molecular interactions are conserved through the evolution (Pagel et al. 2004; Brown and Jurisica 2007). Such algorithm allows to map interactions of model organisms into interactions of human. The database is built starting from known interactions: the algorithms first determine orthologs by using BLASTP, and then an interaction in the model organism is mapped into human if both the interactors have a correspondent ortholog into the human. I2D can be queried by using single or multiple protein identifiers. Results can be visualized by using the I2D graph visualization tool, or by using NAViGaTOR (Network Analysis, Visualization, Graphing TORonto)

(Brown et al. 2009), a visualization tool freely available at (ophid.utoronto.ca/navigator/). Data can be exported both in PSI-MI or tab-delimited formats. I2D is publicly accessible at ophid.utoronto.ca.

IntNetDB

The Integrated Network Database (IntNetDB) (Kai et al. 2006) contains human-predicted protein-protein interactions. The prediction algorithm uses a probabilistic model that exploits 27 datasets containing genomic, proteomic, and functional annotations data. IntNetDB can be queried by inserting protein identifiers through a web interface. Results are visualized in a tabular format or as graphs that may be exported in SVG (Scalable Vector Graphics) or visualized using the SVGviewer. IntNetDB provides further functions for network visualization and analysis.

STRING

The Search Tool for the Retrieval of Interacting Genes/Proteins database (STRING) (von Mering et al. 2005) is a database of predicted interactions with special focus on protein-to-protein, protein-to-DNA, and DNA-to-DNA interactions. The database can be queried by specifying a protein identifier or the protein primary sequence. Results are showed in a prediction summary window and edges of the graph are colored to indicate different types of evidence (fusion evidence, neighborhood evidence, or co-occurrence evidence). STRING is publicly accessible at string-db.org.

Biological Databases on the Cloud

Usually bioinformatics data analysis pipelines include access to public available biological databases that provides web services interfaces (e.g., through the RESTful approach). To improve performance and scalability, such pipelines are deployed on the cloud (Calabrese and Cannataro 2016). To further improve performance when accessing biological databases available on the Internet, such databases are ported on the cloud too. By loading both software tools and databases on the cloud and providing them as a service, it is possible to increase

the level of integration between software, experimental data, and biological databases that improves the analysis and the storage of bioinformatics big data. The provision of data using the data as a service (DaaS) paradigm is thus an emerging approach for the deployment of biological databases. Examples of biological databases deployed on the cloud are contained in the DaaS of the Amazon Web Services (AWS), which provides a centralized repository of public data sets, such as GenBank, Ensembl, 1000 Genomes Project, Unigene, and Influenza Virus (aws.amazon.com/publicdatasets) (Fusaro et al. 2011). The work (Calabrese and Cannataro 2015) presents an overview of main cloud-based bioinformatics systems and discusses challenges and problems related to the use of the cloud in bioinformatics.

Data Analysis Pipelines in Bioinformatics

This section presents main software pipelines for preprocessing and analysis of microarray, mass spectrometry, and interactomics big data.

Analysis of Microarray Data: Gene Expression

The preprocessing of gene expression microarray data can be organized as a pipeline of different tools that aims (i) to identify and remove noise and artifacts related to the microarray experiment, (ii) to extract the correct value of expression for each gene, (iii) to match probes with nucleotide sequences, and (iv) to enrich data through annotations. For each step several algorithms have been developed that may be general purpose or may be designed for a specific chip of the various microarray platforms. Main microarray developers are Affymetrix and Illumina that generate respectively Affymetrix CEL Files and Illumina Tagged Images files. In detail, the workflow for preprocessing and analyzing microarray data comprises four phases (i) preprocessing, which includes summarization and normalization; (ii) annotation, which enriches preprocessed data; (iii) statistical and/or data mining analysis;

and (iv) biological interpretation. Summarization recognizes the position of different genes in raw images, associating some set of pixels to the gene that generated them. Normalization corrects the variation of gene expression in the same microarray due to experimental bias. Filtering reduces the number of investigated genes considering biological aspects, e.g., genes of known functions, or statistical criteria (e.g., associated p-value). Annotation associates to each gene a set of functional information, such as biological processes related to a gene, and cross reference to biological databases. Statistical and data mining analysis aim to identify biologically meaningful genes, e.g., by finding differentially expressed genes among two groups of patients on the basis of their expression values, in case-control studies. In biological interpretation, extracted genes are correlated to the biological processes in which they are involved, e.g., a set of under-expressed genes may be related to the insurgence of a disease. Main challenges of microarray data analysis are discussed in Guzzi and Cannataro (2011). In the following, relevant software tools for gene expression preprocessing and analysis are briefly described.

Affymetrix Power Tools

Affymetrix Power Tools (APT) is a set of command line programs implementing various algorithms for preprocessing both gene expression and SNP Affymetrix microarray data (www.affymetrix.com). Considering gene expression data, the APT command line tool to normalize and summarize an Affymetrix microarray dataset is `apt-probeset-summarize`, and it provides different summarization algorithms, such as the Robust Multi-array Average (RMA) and the Probe Logarithmic Intensity Error (PLIER) algorithms. A possible command line to normalize and summarize an Affymetrix microarray dataset is `apt-probeset-summarize -a rma -d HuEx-1_0-st-v2.cdf -o output -cel-files list.txt`, where `rma` specifies the RMA summarization algorithm, `-d HuEx-1_0-st-v2.cdf` specifies the Human Exon 1.0 st library, `-o output` specifies the output folder, and `-cel-files`

`list.txt` specifies the input folder, where `list.txt` specifies the list of input CEL files. Considering SNP genotyping data, the APT command line tool to preprocess DMET data is `apt-dmet-genotype` that allows the extraction of genotype call (e.g., the individuation of the corresponding nucleotide) for each probeset. Using the `apt-dmet-genotype` command line, it is possible to control main preprocessing parameters, such as the algorithm used to determine the genotype call.

Affymetrix Expression Console

Expression Console is a software tool provided by Affymetrix that implements probe set summarization of Affymetrix binary CEL files and provides both summarization and quality control algorithms. A main advantage of Expression Console is the quality control, but it is not extensible for the preprocessing of multivendor datasets.

Micro-CS

Micro-CS or μ -CS (Microarray CEL file Summarizer) is a software tool for the automatic normalization, summarization, and annotation of Affymetrix gene expression data that adopts a client-server architecture (Guzzi and Cannataro 2010). The μ -CS client is deployed as a plug-in of the TIGR M4 platform or as a Java stand-alone tool. It encloses the Affymetrix Power Tools code and allows users to read, preprocess, and analyze binary microarray gene expression data, providing the automatic loading of preprocessing libraries and the management of intermediate files. The μ -CS server automatically updates a repository of preprocessing and annotation libraries by connecting in a periodic way to the Affymetrix libraries repository. Then, it provides to the μ -CS client the correct and more recent libraries needed for preprocessing the microarray data. The μ -CS server is implemented as a web service. Users can preprocess microarray data without the need to locate, download, and invoke the proper preprocessing tools and chip-specific libraries. Thus, main advantages of using μ -CS are avoiding wasting time for finding correct libraries, reducing errors due to incorrect

parameter settings or for using old libraries, and guarantee of analysis quality by providing the most updated annotation libraries.

dChip

dChip (DNA-Chip Analyzer) is a stand-alone program for summarizing Affymetrix gene expression data. dChip offers both preprocessing and analysis functions, performs the model-based intensity normalization only, and is available for Windows platform only. Compared to μ -CS, dChip does not download automatically the needed libraries nor the annotation of preprocessed data. To preprocess gene expression data with dChip, users need to manually download libraries, to perform the summarization method, and finally to manually download annotation libraries to annotate results. dChip is publicly accessible at www.dchip.org.

RMAExpress

RMAExpress is a stand-alone program for summarizing Affymetrix gene expression data (rma-express.bmbolstad.com). RMAExpress performs only the RMA normalization, is available for Windows platform, and must be compiled for running on the Linux platform. Compared to μ -CS, RMAExpress does not download automatically the needed libraries nor perform the annotation of files. To preprocess gene expression data with RMAExpress, users need to manually download libraries, to perform the summarization method (RMA), and finally to manually download annotation libraries to annotate results.

easyExon

easyExon (Chang et al. 2008) is a software pipeline for preprocessing and analysis of exon array data. easyExon can manage either raw binary CEL files, by transparently calling Affymetrix APT tools, or already summarized files. easyExon integrates external tools such as the Affymetrix Power Tools (APT) for the data preprocessing and the Integrate Genome Browser (IGB) for the biological interpretation of results. A drawback is related to the fact that it focuses only on exon array data.

Analysis of Microarray Data: SNPs

Single nucleotide polymorphisms (SNPs) are a common form of polymorphism in the human genome. Several genotyping platforms for detecting SNPs are widely used for human evolution research and for association studies of complex diseases (Wollstein et al. 2007). Preprocessing and analysis of SNPs aim to identify genetic variations that may be related with diseases. Genome-wide Association Studies (GWAS) investigate all the genome of a large population for finding variations that are statistical significant. The workflow of analysis of GWAS starts with the translation of microarray raw images into the genotype, i.e., the nucleotide sequence. After determining the sequence of genes, statistical models evaluate the significance of the presence or absence of specific variations. GWAS experiments generate big data (e.g., a study may require more than a terabyte of data), and main challenges are the efficient storage and processing of data that require big storage systems and high-performance computational platforms for the analysis. In the following relevant software tools for SNP genotyping data, preprocessing and analysis are briefly described.

DMET Console

The Affymetrix DMET Console is a stand-alone application that analyzes the intensity data (CEL files) from Affymetrix DMET Plus arrays and determines the genotype for each marker. DMET Console generates genotype calls (CHP files) for datasets of intensity data generated by the Affymetrix DMET Plus arrays, and it translates the genotypes to haplotype alleles that are reported using standardized star allele nomenclature. DMET Console tool returns a table containing, for each probeset and for each sample, the detected SNP or NoCall (i.e., the platform has not revealed the nucleotide with a sufficient confidence). The Affymetrix DMET Console (Sissung et al. 2010) is similar to `apt-dmet-genotype`, but it has a graphical user interface and is able to combine all the .CHP files of an experiment to obtain a unique table for all samples.

DMET-Analyzer

The workflow of analysis of a DMET dataset comprises the following steps (Arbitrio et al. 2016b): (i) biological samples are collected and treated to perform microarray experiments; (ii) the DMET microarray experiment is performed producing a raw microarray data file (.CEL file) for each sample; (iii) raw microarray data files are preprocessed by the `apt-dmet-genotype` command line tool, or by the DMET Console software tool, that produce as output respectively a single preprocessed files (.CHP) or a table of SNPs that represent the whole preprocessed dataset; and (iv) discovery of statistically significant differences in SNPs distribution among sample classes, e.g., in case-control studies. DMET-Analyzer (Guzzi et al. 2012) performs the automatic association analysis among the variations in the genomes of patients and their clinical conditions, e.g., a different response to drugs. For each detected SNP, and for each probe, DMET-Analyzer checks the association among the presence of SNPs and the classes, using the statistical Fisher's test. It provides the Bonferroni and false discovery rate (FDR) multiple test corrections, to improve the statistical significance of results. Each significant SNP detected by the DMET platform and underlined by DMET-Analyzer has an entry into dbSNP, allowing scientists to further study interesting SNPs into dbSNP. As the number of patients enrolled in pharmacogenomics studies increases, parallel and cloud computing may be used to face storage and performance issues. In particular, coreSNP (Guzzi et al. 2014) and Cloud4SNP (Agapito et al. 2013a) are, respectively, the parallel- and cloud-based versions of DMET-Analyzer.

Analysis of Mass Spectrometry Data: Protein Identification

Preprocessing and analysis of mass spectrometry proteomics data regard the following main goals: (i) protein identification, i.e., the identification of the amino acid sequence of the proteins contained in a sample; (ii) protein quantification, i.e., the identification of the absolute or relative amount of the proteins contained in a sample; and (iii) biomarker discovery, i.e., the discovery of

discriminant features in spectra able to discriminate samples coming from healthy or diseased people. Mass spectrometry protein identification methods (Perkins et al. 1999; Nesvizhskii 2007) include peptide mass fingerprinting (PMF) from simple mass spectrometry (MS) spectra and protein identification from peptide peaks obtained through tandem mass spectrometry (MS/MS), also called peptide fragmentation fingerprinting (PFF). In PMF, after a specific enzymatic cleavage of a protein and a mass spectrometry measurement of the resulting peptides, the masses of generated peptides form a map in the spectra called peptide mass fingerprinting. Protein identification is obtained by comparing such spectra against a database of masses of theoretical peptides obtained by in silico digestion. In PFF, a peptide/protein (precursor ion) is selected, isolated, and fragmented using tandem mass spectrometry (MS/MS). The fragmentation permits the de novo sequencing, or the masses of the fragments are compared against a database of masses of theoretical peptides obtained by in silico digestion.

MASCOT

MASCOT (Perkins et al. 1999) performs protein identification through PMF and scores results using the MOWSE (MOlecular Weight SEArch) algorithm. Spectra generated by the user are compared against the theoretical spectra, obtained by an enzymatic digestion of protein sequence in a reference database. For each observed peak in a spectrum, the system calculates the probability that a match between a peak and a peptide is purely random. DMET Console is available here www.matrixscience.com.

swissPIT

swissPIT (Swiss Protein Identification Toolbox) (Quandt et al. 2009) runs workflows of analysis of mass spectrometry data on the Swiss BioGRID infrastructure. swissPIT uses different peptide identification tools so the user can submit proteomics data to multiple analysis tools to improve the protein identification process. A single-input data may be submitted to the same tool and analyzed using different parameter settings,

or it can be submitted to different tools. Such computations are executed in parallel on a high-performance computing infrastructure, i.e., the Swiss BioGRID, by using a web-based interface.

EIPeptDi

The ICAT-labeling method allows the identification and quantification of proteins present in two samples. Nevertheless, when the ICAT is coupled to LC-MS/MS, the analysis pipeline presents a drawback: the insufficient sample-to-sample reproducibility on peptide identification. This results in a low number of peptides that are simultaneously identified in different experiments. The EIPeptDi tool (Cannataro et al. 2007) improves peptide identification in a set of spectra, by using a cross validation of peaks commonly detected in different spectra in tandem mass spectrometry. As a result, EIPeptDi increases the overall number of peptides identified in a set of samples.

Analysis of Mass Spectrometry Data: Protein Quantification

Protein quantification aims to infer the relative abundance of proteins. Algorithms include label-based methods that mine data coming from labeled samples, and label-free methods, that analyze the spectra of non-labeled samples. Label-based methods include ASAPRatio, XPRESS, and ProICAT, described in the following, while label-free methods include MSQuant, XCMS, OpenMS, and PeptideProphet.

ASAPRatio

The automated statistical analysis on protein ratio (ASAPRatio) (Li et al. 2003) tool computes the relative abundances of proteins from ICAT data. It preprocesses spectra with a smoothing filter, subtracts background noise from each spectrum, computes the abundance of each couple of corresponding peptides, groups all the peptides that belong to the same protein, and finally computes the abundance of the whole protein.

XPRESS

The XPRESS tool (Han et al. 2001) computes the relative abundance of proteins, from an ICAT-

labeled pair of samples, by reconstructing the light and heavy profiles of the precursor ions and determining.

Analysis of Mass Spectrometry Data: Biomarker Discovery

Biomarker discovery is a key technique for the early detection of diseases and thus for increasing patient survival rates. Biomarker discovery in spectra data aims to find a subset of features of the spectra, i.e., peaks or identified peptide/proteins, able to discriminate the spectra in classes (usually two classes), e.g., spectra belonging to healthy or diseased subjects. Main biomarker discovery algorithms first use spectra preprocessing algorithms to clean and reduce dimension of data, and then apply classification to preprocessed spectra, where classification may be obtained with statistical or data mining methods. A main challenge in biomarker discovery from spectra is the high number of features, i.e., peaks or peptides, compared to the low number of samples.

BioDCV

BioDCV (Cannataro et al. 2007) is a grid-based distributed software for the analysis of both proteomics and genomics data for biomarker discovery. BioDCV solves the selection bias problem that is the discovery of the correct discriminating features, with a learning method based on support vector machines. BioDCV allows to setup complex predictive biomarker discovery experiments on proteomics data.

Trans-Proteomic Pipeline

The Trans-Proteomic Pipeline (TPP) (Deutsch et al. 2010) is a complete suite of software tools, managed by the Seattle Institute of Systems Biology, that supports the complete pipeline of proteomic data analysis. The TPP pipeline contains tools for spectra conversion in standard formats, probabilistic peptide identification, protein quantification, validation, and biological interpretation of results.

Swiss Grid Proteomics Portal

The Swiss Grid Proteomics Portal, now called iPortal, offers several tools for data preprocessing

and analysis and allows to define data analysis workflows combining them (Kunszt et al. 2015). It offers three main portlets (i.e., predefined workflows): quality control, proteomics identification, and proteomics quantification workflows. The portal uses the P-GRADE method to submit workflows to different grid systems or to remote and local resources.

MS-Analyzer

MS-Analyzer is a biomarker discovery distributed platform to classify mass spectrometry proteomics data on the grid (Cannataro and Veltri 2007). It uses ontologies and workflows to support the composition of spectra preprocessing algorithms and data mining algorithms implemented as services. MS-Analyzer implements various spectra preprocessing algorithms and encapsulates several data mining and visualization tools that are classified through two ontologies. The data mining ontology models data analysis and visualization tools, while the proteomics ontology models spectra data and preprocessing algorithms. An ontology-based workflow editor and scheduler allows the easy composition, execution, and monitoring of services on the grid or on a local cluster of workstations. Users can browse the ontologies for selecting the preprocessing and analysis tools suitable to the input spectra data, then compose a workflow using drag and drop functions, and finally execute the workflow on the Grid or on a cluster. The user can design different analysis experiments using a parameter sweep approach (the same workflow is executed in many instances in parallel where each instance has different parameter settings) or using different preprocessing algorithms. This allows to evaluate preprocessing effectiveness in terms of performance (e.g., binned vs not-binned spectra) and in terms of data analysis quality (e.g., classification accuracy when changing the preprocessing type).

Analysis of Protein Interaction Networks: Protein Complexes Prediction

The analysis of the biological properties of protein interaction networks (PINs) can be done

considering the topological properties of the corresponding graph, or by finding relevant substructures in the graph, or comparing the topology of different graphs. Main algorithms analyze the topological properties of a network and extract its global and local properties that are used to infer biological knowledge. For instance, the finding of small subgraphs that are statistically overrepresented may indicate proteins with specific functions. The finding of highly connected subregions may indicate protein complexes. In the following, some tools for the analysis of PINs are described.

The Molecular Complex Detection Algorithm

The molecular complex detection algorithm (MCODE) (Bader and Hogue 2003) was one of the first approaches to extract protein complexes from a protein interaction network. The main idea of MCODE is that protein complexes, i.e., dense sub-networks, are clusters in the graph; thus, it finds complexes by building clusters. MCODE presents three main steps: (i) it weights all vertices considering local network density, where the local area is a subgraph named *k*-core; (ii) then the algorithm uses this resulting weighted graph to extend each subgraph, starting from the highest weighted vertex; When no more vertices can be added to the complex, this task is repeated considering the next highest weighted subgraph; and (iii), finally, protein complexes are filtered considering the number of nodes of each subgraph.

The Markov Cluster Algorithm

The Markov Cluster algorithm (MCL) (Enright et al. 2002) performs graph clustering through the simulation of a stochastic flow in the network and by analyzing the flow distribution. The main idea of MCL is to consider a biological network as a transportation network where there is a random flow. MCL simulates the evolution of the flow as Markov processes, and after a number of iterations, MCL separates network regions with a high flow from regions with low flow.

IMPRECO

IMPRECO (IMproving PREdiction of COMplexes) (Cannataro et al. 2010a) is a parallel tool

for the prediction of protein complexes. It is a meta-predictor; in fact it executes in parallel three prediction algorithms (MCL, MCODE, RNSC) on the input network and then integrates their results trying to predict complexes not predicted by the previous predictors and to improve the overall quality measures of the prediction.

Analysis of Protein Interaction Networks: Local Alignment

As sequence alignment is used to evaluate if similarity among biological sequences (DNA or proteins) is due by chance or by evolution, protein interaction network alignment aims to study the conservation of the interactome of organisms or the conservation of protein complexes among species. Network alignment algorithms comprises two main classes: *global alignment* that finds large common sub-networks covering all the input networks and optimizing a global function based on topology considerations and *local alignment* that finds similar small subregions optimizing a similarity function based on functional similarity, such as homology. Local alignment of interaction networks searches for similar or equal subgraphs in two or more networks (Berg and Lassig 2004). The graph alignment, conceptually similar to the sequence alignment, is based on a scoring function that measures the similarities of two subgraphs that in turn is used to measure the global comparison of interaction networks, by measuring the similarity of mutually similar subgraphs. Local alignment algorithms comparing two or more networks analyze the conservation or divergence of the interaction patterns among different species and give as output a set of conserved subgraphs.

Mawish

Mawish (Koyutürk et al. 2005) is a pairwise local alignment algorithm that finds high similar subgraphs. The algorithm builds an alignment among two networks starting from orthologous nodes. In Mawish the network alignment problem formulates as a graph optimization problem with the aim to build an alignment graph with the maximum score. Mawish includes the following

steps: (i) matching of the orthologs nodes belonging to two networks (a similarity score tables among nodes is built); (ii) growing of the alignment (starting from a pair of similar nodes, the neighborhood of both nodes in the input networks is expanded to small local regions of high similarity using a greedy heuristic); and (iii) evaluation (the aligned subgraphs are evaluated using a statistical approach).

AlignNemo

AlignNemo is a local alignment method that uses and integrates both homology and topology information (Ciriello et al. 2012). AlignNemo finds subgraphs of the networks that have relationship in biological function and in the topology of interactions. The integration algorithm of AlignNemo, which combines homology and topology information, allows better precision and recall performance, compared to other local alignment methods.

AlignMCL

AlignMCL is a local alignment algorithm that finds conserved sub-networks in protein interaction networks (Mina and Guzzi 2012). AlignMCL first builds an alignment graph through the merging of the input protein interaction networks and then mines it to find conserved sub-networks by using the Markov Cluster (MCL) algorithm. AlignMCL is an evolution of AlignNemo, where the mining strategy of AlignNemo, not suitable for big networks, is substituted by the Markov Cluster algorithm that allows better performance. To improve its usability, authors developed a version of AlignMCL as a Cytoscape plug-in that can be used transparently to align protein interaction networks loaded and visualized on Cytoscape.

GLAlign

GLAlign (Global Local Aligner) is a local alignment algorithm that integrates global and local alignment (Milano et al. 2016). The idea of GLAlign is to use the topological information obtained by performing a global alignment of the input networks, to improve the local alignment. In

particular, the information about the global alignment results is used to guide the building of the local alignment. GLAlign uses the MAGNA++ global alignment and the AlignMCL local alignment.

Analysis of Protein Interaction Networks: Global Alignment

Global alignment algorithms find a mapping that involves all the nodes of the input interaction networks. Such algorithms find a relation among all the nodes of the two input networks and may introduce new nodes (dummy nodes) if it is not possible to match some nodes. Opposite to local alignment that finds small regions of similarity or conserved motifs, global alignment finds a mapping that involves all nodes and optimizes a cost function.

MAGNA

At the basis of the MAGNA (Maximizing Accuracy in Global Network Alignment) algorithm (Saraph and Milenković 2014), there is the following consideration about the limits of several global alignment algorithms. In fact, many global alignment methods first compute node similarities to identify the high-scoring alignments with respect to the overall node similarity and then evaluate the accuracy of the alignments by using criteria that consider the number of conserved edges. Thus, they first build the alignment and then verify the number of conserved edges that may be not optimal. Starting from this consideration, MAGNA takes care of edge conservation when the alignment is built and not at the end of the alignment, trying to maintain a good quality of node mapping. The core of MAGNA is a genetic algorithm that simulates a population of alignments that change over time and allows the survival to those alignments that improve accuracy. Other than optimizing the number of conserved edges, MAGNA also optimizes other alignment accuracy measures, such as the combination of node and edge conservation.

MAGNA++

MAGNA++ (Maximizing Accuracy in Global Network Alignment via both node and edge conservation) (Vijayan et al. 2015) extends MAGNA with some important improvements: the possibility to optimize simultaneously edge conservation (with respect to any one of three different measures) and node conservation, improved performance through parallel computing, easy to use graphical user interface, and availability of the source code.

SANA

SANA (Simulated Annealing Network Alignment) is a global network alignment algorithm that uses a new stochastic search algorithm based on simulated annealing (Mamano and Hayes 2017). Authors show that SANA optimizes several popular objective functions improving them with respect to available existing search algorithms. At the basis of the SANA algorithm, there is the consideration that it is difficult to find optimal solutions to the network alignment problem because it is NP-hard. To face the complexity of the problem, authors model the network alignment problem as an optimization problem that is composed by two independent parts: an objective function used to measure the quality the alignment and a search algorithm used to enumerate or find all the possible alignments. The authors also point out that search algorithms may be deterministic or stochastic. The solution to the problem is thus the alignment corresponding to the best value of the objective function.

Visualization of Protein Interaction Networks

Visualization is about the visual drawing of data and is an important analysis step in scientific research. Protein-protein interaction networks are important resources for studying organisms, and their graphic representation may highlight relevant sub-networks or motifs such as protein complexes. However, PIN visualization presents several open problems: very high number of nodes and connections (tens of thousands), heterogeneity of nodes (proteins), heterogeneity of edges (interactions), and availability of

annotations enriching proteins and interactions but complicating their visualization. Many software tools for the visualization of biological networks and specifically PINs have been developed recently. Initial tools were devoted to visualization only, but recent tools offers additional functions for network analysis. The work (Agapito et al. 2013b) analyzes the main software tools for PINs visualization considering these main criteria: technology, interoperability, layout algorithms, rendering algorithms, and available network analysis functions, such as clustering or graph mining. Some tools offer 2D and 3D network visualization through many layout algorithms. Others tools support the integration of network data with annotations. Some tools are dedicated to specific functions such as the study of dynamic aspects of biological processes and analysis and visualization of pathways. An interesting approach is provided by tools like Cytoscape (Yeung et al. 2008) that may be incrementally enriched with novel functions for PIN analysis, through the development of plug-ins by the scientific community. Another trend is the increasing use of parallel computing to improve the performance of the visualization engine that in some cases provides near real-time response time and very high interactivity, as in NAViGaTOR (Brown et al. 2009).

Conclusions

The efficient storage, preprocessing, integration, and analysis of omics and clinical data, is today the main bottleneck of the bioinformatics analysis pipeline. To face those challenges, well-known architectures and algorithms are adapted to the specific issues of big data (e.g., data compression for storage (Cannataro et al. 2001), high-performance distributed computing (Cannataro and Talia 2003; Cannataro et al. 2013), peer-to-peer architectures for improving availability (Cannataro et al. 2008)), or novel computational models are gaining a central role (e.g., Cloud (Calabrese and Cannataro 2015) and RESTful architectures (Agapito et al. 2017), Internet of Things, GPU computing). After introducing big omics data in bioinformatics, the

chapter summarized main biological databases and described main data analysis pipelines available to analyze omics data.

Cross-References

- ▶ [Big Data Analysis Techniques](#)
- ▶ [Big Data Deep Learning Tools](#)
- ▶ [Big Data for Health](#)
- ▶ [Big Data Technologies for DNA Sequencing](#)
- ▶ [Big Data Visualization Tools](#)
- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Databases as a Service](#)
- ▶ [Deep Learning on Big Data](#)
- ▶ [Graph Processing Frameworks](#)
- ▶ [Graph Visualization](#)
- ▶ [Large-Scale Graph Processing System](#)
- ▶ [Parallel Graph Processing](#)
- ▶ [Parallel Processing with Big Data](#)
- ▶ [Privacy-Preserving Data Analytics](#)
- ▶ [Python](#)
- ▶ [Storage Technologies for Big Data](#)
- ▶ [Structures for Large Data Sets](#)
- ▶ [R Language: A Powerful Tool for Taming Big Data](#)
- ▶ [Tools and Libraries for Big Data Analysis](#)
- ▶ [Visualization Techniques](#)
- ▶ [Workflow Systems for Big Data Analysis](#)

References

- Aebersold R, Mann M (2003) Mass spectrometry-based proteomics. *Nature* 422(6928):198–207. <https://doi.org/10.1038/nature01511>
- Agapito G, Cannataro M, Guzzi PH, Marozzo F, Talia D, Trunfio P (2013a) Cloud4SNP: distributed analysis of SNP microarray data on the cloud. In: Proceedings of the international conference on bioinformatics, computational biology and biomedical informatics (BCB'13). ACM, New York, pp 468:468–468:475. <https://doi.org/10.1145/2506583.2506605>
- Agapito G, Guzzi PH, Cannataro M (2013b) Visualization of protein interaction networks: problems and solutions. *BMC Bioinf* 14(1):1
- Agapito G, Guzzi P, Cannataro M (2017) Genotype-panalytic: a restful platform to mine multiple associations between SNPs and drug response in case-control studies. *PeerJ Preprints* 5(e3299v1). <https://doi.org/10.7287/peerj.preprints.3299v1>
- Alfarano C, Andrade CE, Anthony K, Bahroos N, Bajec M, Bantoft K, Bete D, Bobechko B, Boutilier K, Burgess E, Buzadzija K, Cavero R, D'Abreo C, Donaldson I, Dorairajoo D, Dumontie MJ, Dumontier MR, Earles V, Farral R, Feldman H, Garderman E, Gong Y, Gonzaga R, Grytsan V, Gryz E, Gu V, Haldorsen E, Halupa A, Haw R, Hrvojc A, Hurrell L, Isserlin R, Jack F, Juma F, Khan A, Kon T, Konopinsky S, Le V, Lee E, Ling S, Magidin M, Monakis J, Montojo J, Moore S, Muskat B, Ng I, Paraiso JP, Parker B, Pintilie G, Pirone R, Salama JJ, Sgro S, Shan T, Shu Y, Siew J, Skinner D, Snyder K, Stasiuk R, Strumpf D, Tuckam B, Tao S, Wang Z, White M, Willis R, Wolting C, Wong S, Wrong A, Xin C, Yao R, Yates B, Zhang S, Zheng K, Pawson T, Ouellette BF, Hogue CW (2005) The biomolecular interaction network database and related tools 2005 update. *Nucleic Acids Res* 33(Database issue):418–424. http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=pubmed&dopt=Abstract&list_uids=15608229
- Arbitrio M, Di Martino MT, Barbieri V, Agapito G, Guzzi PH, Botta C, Iuliano E, Scionti F, Altomare E, Codispoti S, Conforti S, Cannataro M, Tassone P, Tagliaferri P (2016a) Identification of polymorphic variants associated with erlotinib-related skin toxicity in advanced non-small cell lung cancer patients by dmet microarray analysis. *Cancer Chemother Pharmacol* 77(1):205–209. <https://doi.org/10.1007/s00280-015-2916-3>
- Arbitrio M, Di Martino MT, Scionti F, Agapito G, Guzzi PH, Cannataro M, Tassone P, Tagliaferri P (2016b) DmetTM(drug metabolism enzymes and transporters): a pharmacogenomic platform for precision medicine. *Oncotarget* 7(33):54028
- Bader G, Hogue C (2003) An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinf* 4(1):2. <https://doi.org/10.1186/1471-2105-4-2>, <http://www.biomedcentral.com/1471-2105/4/2>
- Barker W, Garavelli J, Mcgarvey P, Marzec C, Orcutt B, Srinivasarao G, Yeh L, Ledley R, Mewes H, Pfeiffer F, Tsugita A, Wu C (1999) The PIR-international protein sequence database. *Nucleic Acids Res* 27(1):39–43. <http://nar.oxfordjournals.org/cgi/content/abstract/27/1/39>
- Berg J, Lassig M (2004) Local graph alignment and motif search in biological networks. *Proc Natl Acad Sci* 11(101):14689–14694
- Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE (2000) The protein data bank. *Nucleic Acids Res* 28(1):235–242. <https://doi.org/10.1093/nar/28.1.235>
- Boeckmann B, Bairoch A, Apweiler R, Blatter MCC, Estreicher A, Gasteiger E, Martin MJ, Michoud K, O'Donovan C, Phan I, Pilbaut S, Schneider M (2003) The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res* 31(1):365–370. <https://doi.org/10.1093/nar/gkg095>
- Brown K, Jurisica I (2007) Unequal evolutionary conservation of human protein interactions in interologous networks. *Genome Biol* 8(5):R95+. <https://doi.org/10.1186/gb-2007-8-5-r95>

- Brown KR et al (2009) NAViGaTOR: network analysis, visualization and graphing. *Toronto. Bioinformatics* 25(24):3327–3329. <https://doi.org/10.1093/bioinformatics/btp595>, [http://bioinformatics.oxfordjournals.org/content/25/24/3327.full.pdf+html](http://bioinformatics.oxfordjournals.org/content/25/24/3327.abstract)
- Calabrese B, Cannataro M (2015) Cloud computing in healthcare and biomedicine. *Scalable Comput Pract Exp* 16(1):1–18. <http://www.scpe.org/index.php/scpe/article/viewFile/1057/424>
- Calabrese B, Cannataro M (2016) Bioinformatics and microarray data analysis on the cloud. In: Guzzi PH (ed) *Microarray data analysis, methods in molecular biology*, vol 1375. Springer, New York, pp 25–39. https://doi.org/10.1007/7651_2015_236
- Cannataro M (2008) Computational proteomics: management and analysis of proteomics data. *Brief Bioinform bbn011+*. <https://doi.org/10.1093/bib/bbn011>
- Cannataro M, Guzzi P (2012) *Data management of protein interaction networks*. Wiley, New York
- Cannataro M, Talia D (2003) Towards the next-generation grid: a pervasive environment for knowledge-based computing. In: 2003 international symposium on information technology (ITCC 2003), 28–30 Apr 2003, Las Vegas, pp 437–441. <https://doi.org/10.1109/ITCC.2003.1197569>
- Cannataro M, Veltri P (2007) MS-analyzer: preprocessing and data mining services for proteomics applications on the Grid. *Concurr Comput Pract Exp* 19(15):2047–2066. <https://doi.org/10.1002/cpe.1144>
- Cannataro M, Carelli G, Pugliese A, Saccà D (2001) Semantic lossy compression of XML data. In: Proceedings of the 8th international workshop on knowledge representation meets databases (KRDB 2001), Rome, 15 Sept 2001. <http://ceur-ws.org/Vol-45/05-cannataro.pdf>
- Cannataro M, Cuda G, Gaspari M, Greco S, Tradigo G, Veltri P (2007) The EIPeptidi tool: enhancing peptides discovering in icat-based LC MS/MS experiments. *BMC Bioinf* 8:255. <https://doi.org/10.1186/1471-2105-8-255>. Published on-line
- Cannataro M, Talia D, Tradigo G, Trunfio P, Veltri P (2008) Sigmcc: a system for sharing meta patient records in a peer-to-peer environment. *FGCS* 24(3): 222–234. <https://doi.org/10.1016/j.future.2007.06.006>
- Cannataro M, Guzzi PH, Veltri P (2010a) Impreco: distributed prediction of protein complexes. *Futur Gener Comput Syst* 26(3):434–440
- Cannataro M, Guzzi PH, Veltri P (2010b) Protein-to-protein interactions: technologies, databases, and algorithms. *ACM Comput Surv* 43(1). <https://doi.org/10.1145/1824795.1824796>
- Cannataro M, Barla A, Flor R, Jurman G, Merler S, Paoli S, Tradigo G, Veltri P, Furlanello C (2007) A grid environment for high-throughput proteomics. *IEEE Trans Nanobiosci* 6(2):117–123. <https://doi.org/10.1109/TNB.2007.897495>
- Cannataro M, Guzzi PH, Sarica A (2013) Data mining and life sciences applications on the grid. *Wiley Interdisc Rew Data Min Knowl Disc* 3(3):216–238. <https://doi.org/10.1002/widm.1090>
- Chang TY, Li YY, Jen CH, Yang TP, Lin CH, Hsu MT, Wang HW (2008) Easyexon – a java-based gui tool for processing and visualization of affymetrix exon array data. *BMC Bioinf* 9(1):432. <https://doi.org/10.1186/1471-2105-9-432>, <http://www.biomedcentral.com/1471-2105/9/432>
- Ciriello G, Mina M, Guzzi PH, Cannataro M, Guerra C (2012) Alignnemo: a local network alignment method to integrate homology and topology. *PLoS One* 7(6):e38107
- Consortium TU (2010) The universal protein resource (UniProt) in 2010. *Nucleic Acids Res* 38(Suppl 1):D142–D148. <https://doi.org/10.1093/nar/gkp846>
- Craig R, Cortens JP, Beavis RC (2004) Open source system for analyzing, validating, and storing protein identification data. *J Proteome Res* 3(6):1234–1242. <https://doi.org/10.1021/pr049882h>, <http://pubs.acs.org/doi/abs/10.1021/pr049882h>, <http://pubs.acs.org/doi/pdf/10.1021/pr049882h>
- Desiere F, Deutsch EW, King NL, Nesvizhskii AI, Mallick P, Eng J, Chen S, Eddes J, Loevenich SN, Aebersold R (2006) The peptideatlas project. *Nucleic Acids Res* 34(Suppl 1):D655–D658. <https://doi.org/10.1093/nar/gkj040>, http://nar.oxfordjournals.org/content/34/suppl_1/D655.abstract, http://nar.oxfordjournals.org/content/34/suppl_1/D655.full.pdf+html
- Deutsch EW, Mendoza L, Shteynberg D, Farrah T, Lam H, Tasman N, Sun Z, Nilsson E, Pratt B, Prazen B, Eng JK, Martin DB, Nesvizhskii AI, Aebersold R (2010) A guided tour of the trans-proteomic pipeline. *Proteomics* 10(6):1150–1159. <https://doi.org/10.1002/pmic.200900375>
- Di Martino MT, Arbitrio M, Guzzi PH, Leone E, Baudi F, Piro E, Prantero T, Cucinotto I, Calimeri T, Rossi M, Veltri P, Cannataro M, Tagliaferri P, Tassone P (2011) A peroxisome proliferator-activated receptor gamma (pparg) polymorphism is associated with zole-dronic acid-related osteonecrosis of the jaw in multiple myeloma patients: analysis by dmet microarray profiling. *Br J Haematol* 154(4):529–533. <https://doi.org/10.1111/j.1365-2141.2011.08622.x>
- Di Martino MT, Guzzi PH, Caracciolo D, Agnelli L, Neri A, Walker BA, Morgan GJ, Cannataro M, Tassone P, Tagliaferri P (2015) Integrated analysis of microRNAs, transcription factors and target genes expression discloses a specific molecular architecture of hyperdiploid multiple myeloma. *Oncotarget* 6(22):19132
- Enright AJ, Van Dongen S, Ouzounis C (2002) An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res* 30(7):1575–1584
- Fusaro V, Patil P, Gafni E, Dennis P, Wall D, PJ T (2011) Biomedical cloud computing with Amazon web services. *Plos Comput Biol* 7. <https://doi.org/10.1371/journal.pcbi.1002147>
- Guzzi PH, Cannataro M (2010) μ -cs: an extension of the tm4 platform to manage affymetrix binary data. *BMC Bioinf* 11(1):315

- Guzzi PH, Cannataro M (2011) Challenges in microarray data management and analysis. In: 2011 24th international symposium on computer-based medical systems (CBMS). IEEE, pp 1–6
- Guzzi PH, Agapito G, Di Martino MT, Arbitrio M, Tassone P, Tagliaferri P, Cannataro M (2012) Dmet-analyzer: automatic analysis of affymetrix dmet data. *BMC Bioinf* 13(1):258. <https://doi.org/10.1186/1471-2105-13-258>
- Guzzi PH, Agapito G, Cannataro M (2014) coreSNP: parallel processing of microarray data. *IEEE Trans Comput* 63(12):2961–2974. <https://doi.org/10.1109/TC.2013.176>
- Han DK, Eng J, Zhou H, Aebersold R (2001) Quantitative profiling of differentiation-induced microsomal proteins using isotope-coded affinity tags and mass spectrometry. *Nat Biotechnol* 19(10):946–951. <https://doi.org/10.1038/nbt1001-946>
- Hewett M, Oliver DE, Rubin DL, Easton KL, Stuart JM, Altman RB, Klein TE (2002) PharmGKB: the pharmacogenetics knowledge base. *Nucleic Acids Res* 30(1):163–165
- Jones P, Côté RG, Martens L, Quinn AF, Taylor CF, Derache W, Hermjakob H, Apweiler R (2006) PRIDE: a public repository of protein and peptide identifications for the proteomics community. *Nucleic Acids Res* 34(Database issue). <https://doi.org/10.1093/nar/gkj138>
- Kai X, Dong D, Jing-Dong JH (2006) IntNetDB v 1.0 an integrated protein-protein interaction network database generated by a probabilistic model. *BMC Bioinf* 508(7):S1
- Koyutürk M, Grama A, Szpankowski W (2005) Pairwise local alignment of protein interaction networks guided by models of evolution. In: Miyano S, Mesirov JP, Kasif S, Istrail S, Pevzner PA, Waterman MS (eds) RECOMB. Lecture notes in computer science, vol 3500. Springer, pp 48–65
- Kunszt P, Blum L, Hullr B et al (2015) Quantitative profiling of differentiation-induced microsomal proteins using isotope-coded affinity tags and mass spectrometry. *Concurr Comput Pract Exp* 27(2):433–445. <https://doi.org/10.1002/cpe.3294>
- Li Xj, Zhang H, Ranish JA, Aebersold R (2003) Automated statistical analysis of protein abundance ratios from data generated by stable-isotope dilution and tandem mass spectrometry. *Anal Chem* 75(23):6648–6657. <https://doi.org/10.1021/ac034633i>
- Li J, Zhang L, Zhou H, Stoneking M, Tang K (2010) Global patterns of genetic diversity and signals of natural selection for human ADME genes. *Hum Mol Genet*. <https://doi.org/10.1093/hmg/ddq498>, <http://hmg.oxfordjournals.org/content/early/2010/12/02/hmg.ddq498.abstract>, <http://hmg.oxfordjournals.org/content/early/2010/12/02/hmg.ddq498.full.pdf+html>
- Mamano N, Hayes W (2017) SANA: simulated annealing far outperforms many other search algorithms for biological network alignment. *Bioinformatics* 33(14):2156–2164. <https://doi.org/10.1093/bioinformatics/btx090>
- Milano M, Cannataro M, Guzzi PH (2016) Glalign: using global graph alignment to improve local graph alignment. In: Tian T, Jiang Q, Liu Y, Burrage K, Song J, Wang Y, Hu X, Morishita S, Zhu Q, Wang G (eds) IEEE international conference on bioinformatics and biomedicine, BIBM 2016, Shenzhen, 15–18 Dec 2016. IEEE Computer Society, pp 1695–1702. <https://doi.org/10.1109/BIBM.2016.7822773>
- Mina M, Guzzi PH (2012) Alignmc: comparative analysis of protein interaction networks through markov clustering. In: BIBM workshops, pp 174–181
- Nesvizhskii AI (2007) Protein identification by tandem mass spectrometry and sequence database searching. In: Matthiesen R (ed) Mass spectrometry data analysis in proteomics. Methods in molecular biology, vol 367. Humana Press, pp 87–119. <https://doi.org/10.1385/1-59745-275-0:87>
- Pagel P, Mewes H, Frishman D (2004) Conservation of protein-protein interactions – lessons from ascomycota. *Trends Genet* 20(2):72–76. <https://doi.org/10.1016/j.tig.2003.12.007>
- Perkins DN, Pappin DJ, Creasy DM, Cottrell JS (1999) Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis* 20(18):3551–3567
- Quandt A, Masselot A, Hernandez P, Hernandez C, Mafioletti S, Appel RDD, Lisacek F (2009) SwissPIT: an workflow-based platform for analyzing tandem-MS spectra using the Grid. *Proteomics* <https://doi.org/10.1002/pmic.200800207>
- Saraph V, Milenković T (2014) MAGNA: maximizing accuracy in global network alignment. *Bioinformatics* pp btu409+. <https://doi.org/10.1093/bioinformatics/btu409>
- Sherry ST, Ward MH, Kholodov M, Baker J, Phan L, Smigielski EM, Sirotnik K (2001) dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res* 29(1):308–311
- Sissung TM, English BC, Venzon D, Figg WD, Deeken JF (2010) Clinical pharmacology and pharmacogenetics in a genomics era: the DMET platform. *Pharmacogenomics* 11(1):89–103. <https://doi.org/10.2217/pgs.09.154>
- Taylor CF, Hermjakob H, Julian RK, Garavelli JS, Aebersold R, Apweiler R (2006) The work of the human proteome organisation's proteomics standards initiative (HUPO PSI). *OMICS* 10(2):145–151. <https://doi.org/10.1089/omi.2006.10.145>
- Vijayan V, Saraph V, Milenković T (2015) MAGNA++: maximizing accuracy in global network alignment via both node and edge conservation. *Bioinformatics* 31(14):2409–2411. <https://doi.org/10.1093/bioinformatics/btv161>
- von Mering C, Jensen L, Snel B, Hooper S, Krupp M, Foglierini M, Jouffre N, Huynen M, Bork P (2005) String: known and predicted protein-protein associations, integrated and transferred across organisms. *Nucleic Acids Res* 33(Database issue): 433–437

Wollstein A, Herrmann A, Wittig M, Mothnagel M, Franke A, Nurnberg P, Schreiber S, Kravczak M, Hampe J (2007) Efficacy assessment of SNP sets for genome-wide disease association studies. *Nucleic Acids Res* 35:e113. <https://doi.org/10.1093/nar/gkm621>

Yeung N, Cline MS, Kuchinsky A, Smoot ME, Bader GD (2008) Exploring biological networks with cytoscape (2008) Exploring biological networks with cytoscape software. *Curr Protoc Bioinform/editorial board, Andreas D Baxevanis [et al]* Chapter 8. <https://doi.org/10.1002/0471250953.bi0813s23>

Big Data Analysis Techniques

Martin Hahmann

Databases Systems Group, Technische Universität Dresden, Dresden, Germany

Synonyms

[Time series prediction](#)

Definitions

Time series forecasting is the process of making predictions of the future values of a time series, i.e., a series of data points ordered by time, based on its past and present data/behavior.

Overview

Forecasting is part of the broader area of time series analytics. It is a vital and commonly used tool for planning and decision-making. Forecasting fits a model to a given time series and uses it to infer the future values of this time series. For this task, a lot of methods and approaches have been proposed, ranging from simple statistical models to complex and computation intensive machine learning approaches. The biggest challenge in time series forecasting is to identify the optimal combination of model and parameters for a given time series. The process of conducting time series forecasting can be illustrated using the method proposed by Box et al. (2008). Although it was designed for the application of *autoregressive (integrated) moving average (AR(1)MA)* models,

the fundamental approach is widely applicable. The procedure is defined by three stages that are iterated:

1. **Model selection:** Stationarity and seasonality are determined, while autocorrelation and partial autocorrelation plots are used to identify the autoregressive and moving average terms.
2. **Parameter estimation:** In order to find the model coefficients that provide the best fit to the data, sophisticated estimation approaches, e.g., maximum likelihood estimation, are used.
3. **Model checking:** The accuracy of the model is evaluated. If the model does not satisfy the given requirements, the process is restarted at the first stage and tries to identify a better model.

The original process has been extended in recent work (e.g., Makridakis et al. 1998) where a preliminary data preparation stage and a final model application stage have been added. Data preparation transforms and differentiates the data in order to make modeling easier. Model application is the actual computation of forecast values.

Key Research Findings

The topic of time series forecasting is an established field of research offering a variety of modeling techniques. These can be structured into *univariate forecasting methods* and *multivariate forecasting methods*. In addition there exist *techniques for handling incomplete time series* and more recent *machine learning approaches*.

Univariate Forecasting Methods

Univariate forecasting concentrates on individual time series, i.e., predictions are created for one time series, and a model is trained for each time series.

The most simple univariate forecasting method is the so-called *naïve* or *persistence* approach. With this approach, a prediction is equal to the last observed value. In time series notation, $\hat{y}_{T+h} = y_T$, where y_T is the last

observed value of the past data y_1, \dots, y_T and \hat{y}_{T+h} is the forecast for time $T + h$, with forecast horizon h . This method is generally used to provide a baseline for the comparison and evaluation of more sophisticated forecasting models. Another simple technique is the *average* approach, which uses the mean of the past data as prediction $\hat{y}_{T+h} = \bar{y}_T = (y_1 + \dots + y_T)/T$. The *drift* approach uses the average change over time (called drift) in the observed data and adds to the last observed value to create a forecast $\hat{y}_{T+h} = y_t + \frac{h}{T-1} \sum_{t=2}^T (y_t - y_{t-1}) = y_t + h \left(\frac{y_t - y_1}{T-1} \right)$.

Out of the more sophisticated univariate forecasting methods, the *ARIMA* model (McCarthy et al. 2006; Box et al. 2008) is the most well-known. The ARIMA uses three concepts for the modeling of a time series: autoregression **AR**, integration **I**, and moving average **MA**. These different concepts provide the ARIMA model with a high degree of adaptability to different time series characteristics, making it applicable in a wide range of use cases. The modeling begins with the integration step, in which the time series is differentiated to eliminate trend and seasonal characteristics to make it stationary, i.e., its expectation value and variance are constant over time. In the next step, the two predictive model parts AR and MA are applied. The autoregressive part models future time series values based on the most recent historical time series values. The moving average part models future values based on error terms which are the result of the simulated prediction of the available time series history. Both model parts are further distinguished into nonseasonal and seasonal variants. A seasonal model has to be applied if a seasonal pattern is present in the time series. For every analyzed time series, the degree of differentiation has to be determined, and the right number of AR and MA components has to be identified in order to create the optimal ARIMA model. Some guides on how to find the right ARIMA model for a specific time series have been published in Robert Nau (2016) and Box et al. (2008).

Another commonly used univariate technique is exponential smoothing, also known as the Holt-Winters model (McCarthy et al. 2006; Holt 2004). This model uses the exponential window

function to apply a smoothing mechanism on the entire time series. In doing so, exponentially decreasing weights are assigned to all observed values over time, i.e., the farther back in time a value was observed, the smaller its weight. The simplest form of exponential smoothing is $s_t = \alpha \cdot y_t + (1 - \alpha) \cdot s_{t-1}$ where α is the *smoothing factor* with $0 < \alpha < 1$. As such, the smoothed time series is a weighted average of the current observation y_t and the previous smoothed time series s_{t-1} where α controls the smoothing effect and thus the impact of recent changes. To create a forecast, the smoothed statistic is used with the last observed value $\hat{y}_{T+h|T} = \alpha \cdot y_T + (1 - \alpha) \cdot s_{T-1}$. The model is also capable of handling time series with trend (*double exponential smoothing*) and seasonal (*triple exponential smoothing*) characteristics. This is done by introducing additional components that take potential trend and seasonal behavior into account. These components are smoothed themselves and added to the forecast calculation using individual smoothing factors.

Two more recent additions to this class are *multivariate adaptive regression splines* (MARS) (Friedman 1991) and *gradient boosting machines* (GBM) (Friedman 2001). MARS uses a set of external influences of which it automatically extracts the most relevant ones to forecast the currently analyzed time series. Furthermore, it is capable of modeling nonlinear relationships between the external influences and the target series. The GBM model uses an ensemble of weak predictors, typically regression trees, to combine their predictions into the final forecast for the analyzed time series. The input for the decision trees may be historical data of the time series itself or external influences.

Multivariate Forecasting Methods

Multivariate forecasting techniques focus on the analysis and prediction of multiple time series within one model. The most commonly referenced approaches are VARIMA models (Riise and Tjøstheim 1984; Tiao and Box 1981). Like ARIMA, they are based on the concepts of autoregression and moving averages but apply them to a set of time series. In doing so, they model

how the time series in a data set influence each other. Each time series is modeled based on its own past data, the past data of all other time series of the set, and an error term. Assuming a simple two-dimensional VARIMA model with two time series $y_t^{(1)}$ and $y_t^{(2)}$, the model equations are:

$$y_t^{(1)} = c_1 + a_{11}y_{t-1}^{(1)} + a_{12}y_{t-1}^{(2)} + \varepsilon_1$$

$$y_t^{(2)} = c_2 + a_{21}y_{t-1}^{(1)} + a_{22}y_{t-1}^{(2)} + \varepsilon_2$$

where c is a constant value and ε is an error term. The model parameters c and a must be estimated. Econometric models build up on this idea and add external influences which affect the time series that should be predicted but are not affected themselves by other time series (Chatfield 2000).

Techniques for Incomplete Time Series Data

The forecasting of time series requires complete historic data. Gaps and missing values generally prevent the creation of a forecasting model. For this reason, approaches for handling incomplete or intermittent time series have been discussed in the forecasting literature. Croston's method is the most widely used approach for this kind of data and was designed for intermittent time series of nonnegative integer values (Croston 1972; Shewhart and Hyndman 2005). The model involves two exponential smoothing processes. One models the time period between recorded (non-zero) observations of the time series, while the other is used to model the actual observed values. As Croston's method models single time series only, it belongs to the class of univariate techniques. It is mentioned in this separate category because it was especially developed for the context of incomplete time series data. While Croston's method is able to calculate forecasts for incomplete time series, missing values have to occur regularly in order to be properly modeled. This can be problematic in application scenarios, where regular patterns of missing observations are not given.

Another possibility to deal with incomplete time series is to exploit the often hierarchical structure of large data sets (Friedner 2001). For this, time series are transferred to a more coarse

grained aggregation level by the application of an aggregation function, e.g., the sum. For example, time series representing the energy consumption of many individual consumers are aggregated to a higher aggregation level representing a whole city or a certain district. With this, missing observations can be compensated as observation gaps in one time series can be covered by observations from other time series. In addition, the influence of noisy data is reduced as fluctuations of different time series can equalize themselves. However, this approach is not guaranteed to solve the problem of incomplete data entirely. Observation gaps can persist even at high aggregation levels, e.g., if only few time series are available for aggregation. In addition, aggregation leads to a loss of detail and hampers the calculation of forecasts on more fine-grained levels of data.

Machine Learning Approaches

Techniques from the area of machine learning can also be applied to the topic of forecasting. Two well-known representatives are neural networks (Faraway and Chatfield 1998) and support vector regression (Basak et al. 2007). In the context of forecasting, both techniques are mostly used to solve regression problems, i.e., the dependence of a time series value on its predecessors, just as the AR component of the ARIMA model. These approaches can be considered as univariate techniques as they generally focus on single time series. However, they also form a class of their own as they utilize a different modeling approach. While typical univariate techniques model the time series as a whole, machine learning approaches often separate the sequence of observations into tuples containing an influenced historical observation and the observation(s) that influence it. These tuples are then used to train a model which solves the regression problem. This makes machine learning approaches more robust against missing values and also allows some techniques to model nonlinear dependencies in the observed data. On the downside, these methods have a reduced modeling versatility (e.g., no trend or season models) and are computationally expensive due to the higher number of parameters that have to be optimized.

Applications of Time Series Forecasting

Time series forecasting is an important tool for planning and decision-making in many application domains. A typical domain for forecasting is commerce, where predictions of sales quantities are used to observe market development and steer companies accordingly. Over the last decades, renewable energy production with its integration in the energy market has established itself as a major application domain for time series forecasting. Due to the fluctuating nature of renewable energy sources, considerable planning is necessary in order to guarantee stable energy supply and grid operation. A lot of research effort is and has been spent on the development of accurate models for the forecasting of energy demand and energy production of renewable energy sources (Arrowhead 2015; GOFLEX 2017). In recent years, the *Internet of things (IOT)* (Vermesan and Friess 2013) has become another major application domain for time series forecasting. The IOT trend has led to a wide range deployment of embedded sensors into physical devices ranging from home appliances to vehicles and even facilities. All these sensors create time series, and the prediction of these time series is of interest in many domains. For example, the German “Industry 4.0” initiative aims at the integration of IOT data in manufacturing. There, key performance indicators are forecasted in order to plan production and maintenance schedules.

Future Directions for Research

While time series forecasting has been studied extensively, a lot of research continues to be done in this field. One major direction for research is the development of forecasting models for certain application domains. This means that specific domain knowledge is directly integrated into the forecast model in order to improve forecast accuracy. Another major research direction considers the handling and utilization of large-scale time series data. The data gathering trend in general and IOT in particular have made large sets of re-

lated time series data (e.g., smart meters installed in the same city, pressure sensors on multiple tools of the same type) more and more available. Thus, the integration of such “peer” time series into multivariate forecasting models has become an important research question.

Cross-References

- [Apache Mahout](#)
- [Apache SystemML](#)
- [Big Data Analysis and IOT](#)
- [Business Process Analytics](#)
- [Business Process Monitoring](#)

References

- Arrowhead (2015) Arrowhead framework, 28 Apr 2015. <http://www.arrowhead.eu/>
- Basak D, Pal S, Patranabis DC (2007) Support vector regression. *Neural Inf Process Lett Rev* 11(10):203–224. <https://doi.org/10.4258/hir.2010.16.4.224>
- Box GEP, Jenkins GM, Reinsel GC (2008) Time series analysis: forecasting and control. Wiley series in probability and statistics. Wiley, Hoboken
- Chatfield C (2000) Time-series forecasting. Chapman & Hall/CRC Press, Boca Raton
- Croston JD (1972) Forecasting and stock control for intermittent demands. *Oper Res Q* 23:289–303
- Faraway J, Chatfield C (1998) Time series forecasting with neural networks: a comparative study using the airline data. *J R Stat Soc Ser C Appl Stat* 47(2): 231–250
- Fliedner G (2001) Hierarchical forecasting: issues and use guidelines. *Ind Manag Data Syst* 101(1):5–12. <https://doi.org/10.1108/02635570110365952>
- Friedman JH (1991) Multivariate adaptive regression splines. *Ann Stat* 19(1):1–67. <https://doi.org/10.1214/aos/1176347963>
- Friedman JH (2001) Greedy function approximation: a gradient boosting machine. *Ann Stat* 29(5):1189–1232
- GOFLEX (2017) GOFLEX project, 11 Feb 2017. <http://goflex-project.eu/>
- Holt CC (2004) Forecasting seasonals and trends by exponentially weighted moving averages. *Int J Forecast* 20(1):5–10. <https://doi.org/10.1016/j.ijforecast.2003.09.015>
- Makridakis SG, Wheelwright SC, Hyndman RJ (1998) Forecasting: methods and applications. Wiley, New York
- McCarthy TM, Davis DF, Golicic SL, Mentzer JT (2006) The evolution of sales forecasting management:

- a 20-year longitudinal study of forecasting practices. *J Forecast* 25(5):303–324. <https://doi.org/10.1002/for.989>
- Riise T, Tjostheim D (1984) Theory and practice of multivariate arma forecasting. *J Forecast* 3(3):309–317. <https://doi.org/10.1002/for.3980030308>
- Robert Nau (2016) Statistical forecasting: notes on regression and time series analysis. <http://people.duke.edu/~rnau/411home.htm>. Accessed 09 Nov 2016
- Shenstone L, Hyndman RJ (2005) Stochastic models underlying Croston's method for intermittent demand forecasting. *J Forecast* 24(6):389–402
- Tiao GC, Box GEP (1981) Modeling multiple time series with applications. *J Am Stat Assoc* 76(376):802–816. <https://doi.org/10.1080/01621459.1981.10477728>
- Vermesan O, Friess P (2013) Internet of things: converging technologies for smart environments and integrated ecosystems. River Publishers, Aalborg

Big Data Analytics

- Big Data in Social Networks

Big Data and Exascale Computing

Amelie Chi Zhou¹ and Bingsheng He²
¹Shenzhen University, Shenzhen, China
²National University of Singapore, Singapore, Singapore

Definitions

Big data refers to data sets which have large sizes and complex structures. The data size can range from dozens of terabytes to a few petabytes and is still growing. Big data is usually characterized with three Vs, namely the large Volume of data size, high Variety of data formats and fast Velocity of data generation.

The importance of big data comes from the value contained in the data. However, due to the three Vs, it is challenging to obtain the value from big data fastly and efficiently. Thus, designing big data systems which analyze big data according to the features of the data is important.

Overview

In the past few decades, clusters, grids, and cloud computing systems have been used to address the challenges on big data processing. Those systems use large numbers of commodity machines interconnected using commodity Ethernet networks to improve system scalabilities horizontally. Various programming models have also been proposed to efficiently distribute big data processing jobs onto those systems, such as MapReduce (Dean and Ghemawat 2004) and Apache Beam (Apache 2017). Based on the programming models, many big data processing systems have been designed to execute and manage data processing jobs with different features in parallel, such as Hadoop (Apache 2011), Yarn (Vavilapalli et al. 2013), and Spark (Zaharia et al. 2010) for batch jobs; Spark Streaming (Zaharia et al. 2013), Storm (Toshniwal et al. 2014), and Flink (Friedman and Tzoumas 2016) for stream jobs; and Apache Beam for both batch and stream jobs.

On the other hand, HPC communities have been developing powerful supercomputers. Exascale computing has already been on the schedule. As the computational requirement of big data applications increases, researchers envision that exascale computing and big data will converge and interact with each other (Reed and Dongarra 2015).

Exascale Computing

Exascale computing refers to computing systems capable of at least one exafLOPS, i.e., 10^{18} floating point calculations per second. The need of going to exascale becomes imperative with the emerging of big data applications. Especially for extreme-scale scientific applications such as climate modeling (Fu et al. 2017b) and earthquake simulation (Fu et al. 2017a), it can easily take years to complete one simulation if executed on a single core, not to imagine that scientists usually need to run the same simulation for multiple times to evaluate the impact of different parameters.

However, going toward exascale computing is not easy, and the challenges come from both hardware and software aspects. From the hardware side, with the millions of cores in one computer system, the power consumption and resilience problem have become especially important. From the software side, several major challenges must be resolved, such as achieving extreme parallelism to utilize the millions of cores efficiently, energy efficiency, fault tolerance, and I/O efficiency for data-intensive computations.

Current Efforts Toward Exascale Computing

Researchers have made great efforts to address the hardware and software challenges of exascale computing.

Energy efficiency. The US Department of Energy (DOE) has set a power budget of no more than 20 MW for an exaflop, forcing co-design innovations in both hardware and software to improve the energy efficiency of exascale computing. On the hardware side, many-core processor design gets more popular in exascale computing to increase the ratio of computational power to power consumption. New technologies such as Intel's Running Average Power Limit (RAPL) are introduced to monitor and control various power and thermal features of processors. The energy efficiency of other hardware components, such as memory and interconnect, is also enhanced to improve the overall efficiency of the systems. On the software side, simulation codes and parallel programming models need to be modified accordingly to better adapt to the new architectural features of future exascale computing systems.

Resilience and fault tolerance. With the large system scale, failures due to hardware and software reasons have become the norm rather than the exception in exascale computing systems. Current efforts have been made on both hardware and software aspects to solve the resilience problem. On the hardware side, system-on-chip designs are proposed to integrate hardware components onto a single chip to reduce the hard error rate. Techniques have also been introduced

to reduce the error rate of each individual component on chip, such as adding redundant cores in processors and using powerful error correction codes in unreliable message channels to reduce the likelihood of failures/errors. On the software side, debugging in exascale is extremely hard as some scalability bugs occur only intermittently at full scale. The current solution is mainly using application-based checkpoint/restart technique (Cappello et al. 2014). Another solution is again introducing redundancy in the software and using replication to reduce the likelihood of software failures. Emerging nonvolatile memory can also be used to reduce the checkpointing overhead (Gao et al. 2015).

The current efforts on developing high-performance computers (HPC) have led to a number of petascale computers. Since 1993, the TOP500 project lists the world's 500 most powerful nondistributed computer systems twice a year. The ranking is mainly based on measuring a portable implementation of the high-performance LINPACK benchmark. Currently, China's Sunway TaihuLight HPC holds the No. 1 position in the TOP500 ranking (released in November 2017), and it reaches 93.015 petaFLOPS on the LINPACK benchmarks and has a theoretical peak performance of 125.436 petaFLOPS. This means that to reach exascale computing, we still need to increase the computation power of existing HPC systems by at least ten times. Table 1 shows the specifications of the first five supercomputers on the latest TOP500 list.

Many countries are planning to have their exascale computers in the next 5 years. For example, according to the national plan for the next generation of high-performance computers, China will have their first operational exascale computer by 2020. The US Exascale Computing Project plans to have capable exascale systems delivered in 2022 and deployed in 2023.

Convergence of Big Data and Exascale Computing

Although the powerful computation capability of supercomputers seems a perfect fit for big data

Big Data and Exascale Computing, Table 1 Specifications of the top five supercomputers in the world (released in November 2017)

Rank	Name	Model	Processor	Interconnect	Vendor	Country	Year
1	Sunway TaihuLight	Sunway MPP	SW26010	Sunway	NRCPC	China	2016
2	Tianhe-2	TH-IVB-FEP	Xeon E52692, Xeon Phi 31S1P	TH Express-2	NUDT	China	2013
3	Piz Daint	Cray XC50	Xeon E5-2690v3, Tesla P100	Aries	Cray	Switzerland	2016
4	Gyoukou	ZettaScaler-2.2 HPC system	Xeon D-1571, PEZY-SC2	Infiniband EDR	ExaScaler	Japan	2017
5	Titan	Cray XK7	Opteron 6274, Tesla K20X	Gemini	Cray	United States	2012

processing, there's a big gap between the two. This is mainly because the design of HPC aims for vertical scaling (i.e., scale-up which means adding more resources to individual nodes in a system), while many big data processing systems require good horizontal scalability (i.e., scale-out which means adding more nodes in a system). Research on how to fill the gap and to lead to the convergence between the two fields is necessary.

Current Efforts Toward Convergence

Various projects and workshops have been proposed to foster the convergence. For example, the BigStorage (EU 2017) European project aims to use storage-based techniques to fill the gap between existing exascale infrastructures and applications dealing with large volume of data. The Big Data and Extreme Computing (BDEC) workshop is jointly organized by over 50 universities, research institutes, and companies in the United States, the European Union, China, and Japan. It started from 2013 and has been looking for pathways toward the convergence between big data and extreme computing. Technically, there are mainly two pathways to prompt the convergence, namely, introducing new exascale architecture compatible for existing big data analytics softwares and introducing new data analytics software for exascale computing platforms.

On the first pathway, current cloud providers are offering high-performance instance types with high-performance processors, lots of memory, and fast networking for computation- and data-intensive applications. The underlying

infrastructures of these instance types are also different from traditional ones which are mostly based on commodity machines. For example, it is recently announced that Cray will bring its physical supercomputers into Windows Azure cloud, for users to easily get access to HPC in cloud.

On the second pathway, scientists are improving the parallelism of data-intensive simulations such as earthquake simulations to better utilize the power of supercomputers. As I/O is important to the overall performance of big data in HPC, new components such as Burst Buffer are introduced in Parallel File System to improve the I/O efficiency of big data in HPC. The traditional workloads in HPC systems are mostly write-intensive, while many big data applications are read-intensive. Burst Buffer has been adopted in storage systems as a prefetcher for read-intensive data analytics jobs to optimize their I/O performances.

Challenges and Opportunities

We see a clear trend and need of the convergence between big data and exascale computing. On the one hand, HPC cloud is emerging in major cloud providers such as Windows Azure, Amazon EC2, and Tencent Cloud to provide high-performance computing resources or even exclusive physical resources for data-intensive applications. On the other hand, new softwares such as Burst Buffer and I/O scheduling algorithms are proposed for optimizing big data applications on the traditional parallel supercomputers. Although these efforts

have offered great opportunities for the convergence, challenges also exist.

Challenges on converging big data and exascale computing mainly lie in the exascale software and hardware co-design. Currently, most efforts on improving the scalability of large-scale simulations in HPC systems are hardware-specific, namely, the optimizations for improving the parallelism of simulation codes are designed for a specific HPC hardware architecture (e.g., Fu et al. 2017a,b). Due to the architectural difference of HPC systems (refer to Table 1), a portable interface for building big data analytics jobs and automatically parallelizing them on HPC systems is needed.

References

- Apache (2011) The Apache Hadoop project. <http://www.hadoop.org>. Accessed 1 Dec 2017
- Apache (2017) Apache Beam: an advanced unified programming model. <https://beam.apache.org/>. Accessed 1 Dec 2017
- Cappello F, Al G, Gropp W, Kale S, Kramer B, Snir M (2014) Toward exascale resilience: 2014 update. *Supercomput Front Innov: Int J* 1(1):5–28
- Dean J, Ghemawat S (2004) Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design & implementation, OSDI'04, vol 6. USENIX Association, Berkeley, pp 137–149
- EU (2017) Bigstorage European project. <http://bigstorage-project.eu/>. Accessed 1 Dec 2017
- Friedman E, Tzoumas K (2016) Introduction to Apache flink: stream processing for real time and beyond, 1st edn. O'Reilly Media Inc., Sebastopol
- Fu H, He C, Chen B, Yin Z, Zhang Z, Zhang W, Zhang T, Xue W, Liu W, Yin W, Yang G, Chen X (2017a) 18.9Pflopss nonlinear earthquake simulation on sunway taihulight: enabling depiction of 18-hz and 8-meter scenarios. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, SC '17. ACM, New York, pp 2:1–2:12
- Fu H, Liao J, Ding N, Duan X, Gan L, Liang Y, Wang X, Yang J, Zheng Y, Liu W, Wang L, Yang G (2017b) Redesigning cam-se for peta-scale climate modeling performance and ultra-high resolution on sunway taihulight. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, SC '17. ACM, New York, pp 1:1–1:12
- Gao S, He B, Xu J (2015) Real-time in-memory checkpointing for future hybrid memory systems. In: Proceedings of the 29th ACM on international conference on supercomputing, ICS '15. ACM, New York, pp 263–272. <http://doi.acm.org/10.1145/2751205.2751212>
- Reed DA, Dongarra J (2015) Exascale computing and big data. *Commun ACM* 58(7):56–68. <http://doi.acm.org/10.1145/2699414>
- Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm@twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD '14. ACM, New York, pp 147–156
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeeschwieler E (2013) Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing, SOCC '13. ACM, New York, pp 5:1–5:16
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing (HotCloud'10). USENIX Association, Berkeley, pp 1–7
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP '13. ACM, New York, pp 423–438

Big Data and Fog Computing

Yogesh Simmhan
Department of Computational and Data Sciences, Indian Institute of Science (IISc), Bangalore, India

Synonyms

Cloudlets

Definitions

Fog Computing A model of distributed computing comprising of virtualized, heterogeneous, commodity computing and storage resources for hosting applications, analytics, content, and services that are accessible with low latency from the edge of wide area networks where clients are present while also having back-end connectivity to cloud computing resources.

Overview

Fog computing is an evolving computing paradigm that offers resources that lie between the edge devices and the cloud data center. It is motivated by the need to process high speed and large volumes of Big Data from Internet of Things with low latency. This article discusses the characteristics Fog computing, data-driven applications on it, and future research problems.

Background

Ever since computers could connect over a network, computing paradigms have undergone cyclical phases on *where* within the network the computation is performed. While the original mainframes till the 1970s were large, centralized time-sharing machines accessed by multiple users through remote terminals, the personal computers (PCs) of the 1980s heralded local processing for individuals (Nelson and Bell 1986). The growth of local area networks (LAN), the Internet, and the World Wide Web (WWW) brought about client-server models in the 1990s where many clients could access content hosted on individual servers, though most of the processing was still done on the PC (Sinha 1992). Web services and e-commerce of the 2000s lead to the growth of *cloud computing*, where computation once again skewed to centralized data centers, but with server farms rather than single servers hosting services that were consumed by PCs (Hwang et al. 2013). A complementary phenomenon in that decade was peer-to-peer (P2P) systems where PCs distributed across the Internet would work collaboratively for content sharing, to tackle bandwidth limitations of the Internet (Milojicic et al. 2002). Both cloud computing and P2P signaled the arrival of the *Big Data* age, where the ability to collect large enterprise, scientific and web datasets, and media content put an emphasis on being able to share and process them at large scales.

The decade of 2010 is seeing a similar cyclical shift but at a faster pace due to several technology advances. Starting with a more centralized cloud

computing model hosting thousands of virtual machines (VMs), we have seen the rollout of pervasive broadband Internet and cellular network communication combined with the rapid growth of smartphones as general-purpose computing platforms backed by cloud computing. *Internet of Things (IoT)* is yet another paradigm, enabled by the convergence of these other technologies (Gubbi et al. 2013). Sensors and constrained devices connected to the Internet are being deployed to support vertical IoT domains such as personal fitness using wearables, smart utilities using metering infrastructure, and even self-driving cars. Both smartphones and IoT mark the advent of *Edge Computing* (or mobile cloud computing). Here, ubiquitous devices numbering in the billions are present at the edge of the wide area network (WAN) that is the Internet and host applications that either operate locally or serve as lightweight clients that publish data to or consume content from cloud services (Garcia Lopez et al. 2015; Fernando et al. 2013).

The limitations of an edge-only or a cloud-only model were recognized by Satyanarayanan et al., who introduced the concept of *cloudlets* (Satyanarayanan et al. 2009). These are resource-rich servers, relative to edge devices, that could host VMs while also being closer in the network topology to the edge devices, relative to cloud data centers. They are designed to overcome the constrained resources available on edge platforms while reducing the network latency expended in being tethered to the cloud for interactive applications. *Fog computing* generalizes (and popularizes) the notion of cloudlets.

The term “fog computing” was coined by Cisco and first appears publicly in a talk by Flavio Bonomi, Vice President and Fellow at Cisco Systems, as part of the *Network-Optimized Computing at the Edge Of the Network Workshop*, colocated with International Symposium on Computer Architecture (ISCA), in 2011 (Bonomi 2011). This was further described as extending the concept of cloud computing to the edge of the network to support low-latency and geodistributed applications for mobile and IoT domains (Bonomi et al. 2014). Since then, fog

computing has been evolving as a concept and covers a wide class of resources that sit between the edge devices and cloud data centers on the network topology, have capacities that fall between edge devices and commodity clusters on clouds, and may be managed ad hoc as a smart gateway or professionally as a computing infrastructure (Varshney and Simmhan 2017; Vaquero and Rodero-Merino 2014; Yi et al. 2015).

Motivations

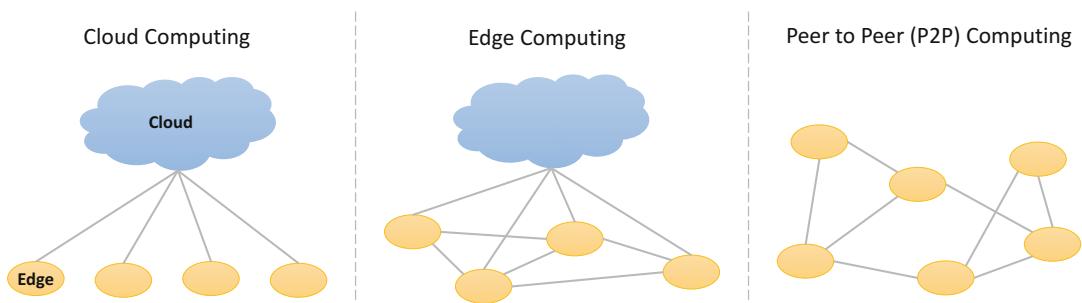
Fog computing is relevant in the context of *wide-area distributed systems*, with many clients at the edge of the Internet (Donnet and Friedman 2007). Such clients may be mobile or consumer devices (e.g., smartphone, smart watch, virtual reality (VR) headset) used interactively by humans or devices that are part of IoT (e.g., smart meter, traffic cameras and signaling, driverless cars) for machine-to-machine (M2M) interactions. The client may serve both as *consumers* of data or actuators that receive control signals (e.g., fitness notification on smart watch, signal change operation on traffic light), as well as *producers* of data (e.g., heart rate observations from smart watch, video streams from traffic cameras) (Dastjerdi and Buyya 2016).

There are two contemporary models of computing for applications and analytics that use data from, or generate content and controls for, such clients at the edge of the Internet (Simmhan 2017), as illustrated in Fig. 1. In a *cloud-centric model*, also called cloud computing, data from these clients is sent to a central data center where the processing is done and the

responses, if any, are sent back to the same or different client(s) (Simmhan et al. 2013; Cai et al. 2017). In an *edge-centric model*, or edge computing, part (or even all) of the processing is done at the data source with the rest done on the cloud (Beck et al. 2014; Anand et al. 2017).

These have their relative advantages. Cloud computing outsources computing infrastructure management to providers, who offer elastic access to seemingly infinite compute resources on-demand which can be rented by the minute. They are also cheaper due to economies of scale at centralized locations (Cai et al. 2017). Edge computing leverages the compute capacity of existing captive devices and reduces network transfers, both of which lower the costs. There may also be enhanced trust and context available closer to the edge (Garcia Lopez et al. 2015).

While fog computing is still maturing, there are many reason why its rise is inevitable due to the gaps in these two common computing approaches. The *network latency* from the edge client to the cloud data center is high and variable, averaging between 20 and 250 ms depending on the location of the client and data center (He et al. 2013). The *network bandwidth* between the edge and the cloud, similarly, averages at about 800–1200 KB/s. Both these mean that latency-sensitive or bandwidth-intensive applications will offer poor performance using a cloud-centric model due to the round-trip time between edge and cloud (Satyanarayanan et al. 2009, 2015). Another factor is the connectivity of devices to the Internet. Mobile devices may be out of network coverage periodically and cause



Big Data and Fog Computing, Fig. 1 Connectivity between resources in cloud, edge, and P2P models

cloud-centric applications to degrade or loose functionality (Shi et al. 2016).

Edge computing while avoiding issues of network time suffers from operating on constrained devices that have limited battery, compute, and memory capacities (Barbera et al. 2013). This reduces application performance and limits sustained processing that can drain the battery. These devices may also be less robust and their network connectivity less available (Aral and Brandic 2017).

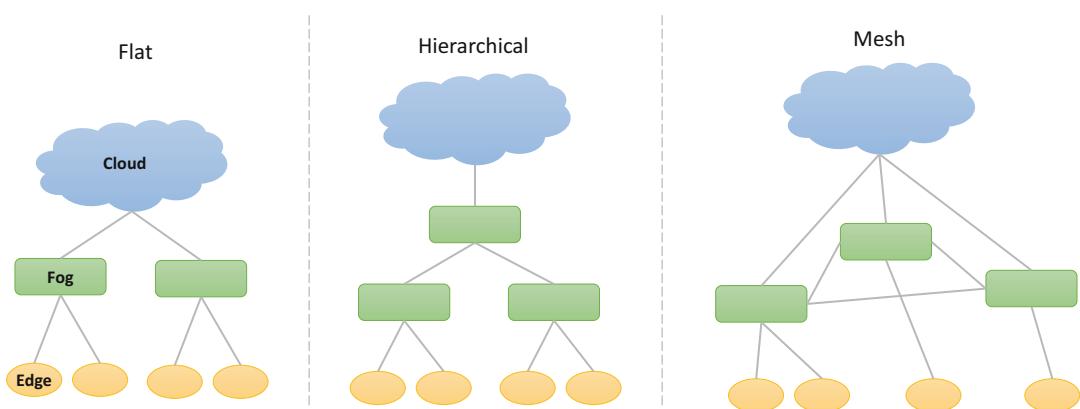
Fog computing serves as a computing layer that sits between the edge devices and the cloud in the network topology. They have more compute capacity than the edge but much less so than cloud data centers. They typically have high uptime and always-on Internet connectivity. Applications that make use of the fog can avoid the network performance limitation of cloud computing while being less resource constrained than edge computing. As a result, they offer a useful balance of the current paradigms.

Characteristics

Fog computing is an evolving model of computing. Hence, its perceived characteristics are broad, based on diverse viewpoints of the research community and industry (Varshney and Simmhan 2017). It is generally agreed that a defining characteristic of fog computing is its smaller network distance from the edge, but mul-

iple **network topology architectures** exist, as shown in Fig. 2 (Aazam and Huh 2014; He et al. In Press; Hou et al. 2016). Some models consider there to be a *flat* fog layer that sits, say, at 1-hop from the edge devices and allows connectivity to the cloud. Each fog device may be responsible for a collection of edge devices that connect to it. This zone of responsibility may be based on logical grouping or spatial regions. Others consider a *hierarchical* approach where fog devices form layers, where each layer is further away from the edge device and responsible for fog/edge devices that fall within its sub-tree, with the cloud forming the root. Alternatively, there are *mesh* designs where fog resources can communicate with each other as well as the cloud, with each edge device assigned to a single fog. A variation of this does not distinguish between edge and fog devices, and any of these may communicate with each other or the cloud. This approaches an edge-centric or P2P model.

The **resource capacity** in the fog can vary. At one end, Raspberry Pi devices with 1 GHz processors, 1 GB RAM, and 100 Mbps Ethernet may serve as a gateway fog resource for lower-end edge devices. On the other hand, fog resources could be provisioned as “micro” or “nano” data centers with clusters of servers or even accelerators present (Garcia Lopez et al. 2015). Individual fog devices can also have heterogeneous capacities (Chiang and Zhang 2016).



Big Data and Fog Computing, Fig. 2 Different interaction models in fog computing

Fog resources tend to be more **reliable and available** than edge resources, though lacking the robust fail-over mechanisms in the cloud that is possible due to a large resource pool. This makes them well-suited to serve as a layer for persisting data and services for the short and medium term. Fog resources themselves may be deployed in a stationary environment (e.g., coffee shop, airport, cell tower) or in a mobile platform (e.g., train, cab) (Chiang and Zhang 2016). This can affect the network connectivity of the fog with the cloud, in case it uses cellular networks for Internet access, and even its energy footprint (Jalali et al. 2017).

Fog **deployment models** are still emerging. These resources may be deployed within a public or a private network, depending on its end use. Smart city deployments may make them available to utility services within the city network (Yannuzzi et al. 2017), while retail shops and transit services can make them available to their customers. A wider deployment for public use on-demand, say, by cellular providers, cities, or even cloud providers, will make it comparable to cloud computing in terms of accessibility. These have implications on the operational costs as well (Viquer and Rodero-Merino 2014).

The fog resources may be made available as-a-service, similar to cloud resources. These may be virtualized or non-virtualized infrastructure (Bittencourt et al. 2015), with containers offering a useful alternative to hypervisor-based VMs that may be too heavyweight for lower-end fog resources (Anand et al. 2017). However, there is still a lack of a common platform, and programming models are just emerging (Hong et al. 2013; Ravindra et al. 2017). Most applications that use the fog tend to be custom designed, and there has only been some theoretical work on scheduling applications on edge, fog, and cloud (Brogi and Forti 2017; Ghosh and Simmhan In Press).

Role of Big Data

One of the key rationales for deploying and using fog computing is Big Data generated at the

edge of the network. This is accelerated by IoT deployments. Traditional web clients which just consume services and content from the WWW saw the growth of *content distribution networks (CDN)* to serve these with low latency. The data from IoT sensors is instead generating data at the clients that is being pushed to the cloud (Cai et al. 2017). In this context, fog computing has been described as acting like an inverse CDN (Satyanarayanan et al. 2015).

A large swathe of IoT data comes as *observational streams*, or time-series data, from widely distributed sensors (Naas et al. 2017; Shukla et al. 2017). These data streams vary in their rates – once every 15 min for smart utility meters, every second for heart rate monitoring by a fitness watch to 50 Hz by phasor measurement units (PMU) in smart power grids – and the number of sensors can range in the millions for city-scale deployments. These are **high-velocity** data streams that are latency sensitive and need online analytics and decision-making to provide, say, health alerts or manage the power grid behavior (Simmhan et al. 2013). Here, fog computing can help move the decision-making close to the edge to reduce latency.

Another class of **high-volume** data that is emerging is from video streams from traffic and surveillance cameras, for public safety, intelligent traffic management, and even driverless cars (Aazam et al. 2016). Here, the bandwidth consumed in moving the data from the edge to the cloud can be enormous as high-definition cameras become cheap but network capacity growth does not keep pace (Satyanarayanan et al. 2015). The applications that need to be supported can span real-time video analytics to just recording footage for future use. Fog computing can reduce the bandwidth consumed in the core Internet and limit data movement to the local network. In addition, it can offer higher compute resources and accelerators to deploy complex analytics as well.

In addition, **telemetry data** from monitoring the health of the IoT fabric itself may form a large corpus (Yannuzzi et al. 2017). Related to this is provenance that describes the source and processing done on the distributed devices that may be

essential to determine the quality and veracity of the data. Fog can help with the collection and curation of such ancillary data streams as well.

Data archival is another key requirement within such applications (Cai et al. 2017; Vaquero and Roderer-Merino 2014). Besides online analytics, the raw or preprocessed observational data may need to be persisted for medium or long term to train models for machine learning (ML) algorithms, for auditing to justify automated decisions, or to analyze on-demand based on external factors. Depending on the duration of persistence, data may be buffered in the fog either transiently or for movement to the cloud during off-peak hours to shape bandwidth usage. Data can also be filtered or aggregated to send only the necessary subset to the cloud.

Lastly, **metadata** describing the entities in the ecosystem will be essential for information integration from diverse domains (Anand et al. 2017). These can be static or slow changing data or even complex knowledge or semantic graphs that are constructed. They may need to be combined with real-time data to support decision-making (Zhou et al. 2017). The fog layer can play a role in replicating and maintaining this across distributed resources closer to the edge.

There has been rapid progress on Big Data platforms on clouds and clusters, with frameworks like Spark, Storm, Flink, HBase, Pregel, and TensorFlow helping store and process large data volumes, velocities, and semi-structured data. Clouds also offer these platforms as a service. However, there is a lack of programming models, platforms, and middleware to support various processing patterns necessary over Big Data at the edge of the network that can effectively leverage edge, fog, and cloud resources (Pradhan et al. In Press).

Examples of Applications

The applications driving the deployment and need for fog computing are diverse. But some requirements are recurrent: low latency processing, high volume data, high computing or storage needs, privacy and security, and

robustness. These span virtual and augmented reality (VR/AR) applications and gaming (Yi et al. 2015), Industrial IoT (Chiang and Zhang 2016), and field support for the military (Lewis et al. 2014). Some of the emerging and high-impact applications are highlighted below.

Smart Cities

Smart cities are a key driver for fog computing, and these are already being deployed. The Barcelona city's "street-side cabinets" offer fog resources as part of the city infrastructure (Yannuzzi et al. 2017). Here, audio and environment sensors, video cameras, and power utility monitors are packaged alongside compute resources and network backhaul capacity as part of fog cabinets placed along streets. These help aggregate data from sensors, perform basic analytics, and also offer WiFi hot spots for public use. As an example, audio analytics at the fog helps identify loud noises that then triggers a surveillance camera to capture a segment of video for further analysis. Similar efforts are underway at other cities as well (Amruthur et al. 2017). These go toward supporting diverse smart city applications for power and water utilities, intelligent transport, public safety, etc., both by the city and by app developers.

One of the major drivers for such city-scale fog infrastructure is likely to be *video surveillance* that is starting to become pervasive in urban spaces (Satyanarayanan et al. 2015). Such city or even crowd-sourced video feeds form *meta-sensors* when combined with recent advances in deep neural networks (DNN). For example, feature extraction and classification can count traffic and crowds, detect pollution levels, identify anomalies, etc. As such, they can replace myriad other environment sensors when supported by real-time analytics. Such DNN and ML algorithms are computationally cost to train and even infer and can make use of fog resources with accelerators (Khochare et al. 2017).

Healthcare

Wearables are playing a big role in not just personal fitness but also as assistive technologies in healthcare. Projects have investigated the use

of such on-person monitoring devices to detect when stroke patients have fallen and need external help (Cao et al. 2015). Others use eye-glass cameras and head-up displays (HUDs) to offer verbal and cognitive cues to Alzheimer's patients suffering from memory loss, based on visual analytics (Satyanarayanan et al. 2009). Predictive analytics over brain signals monitored from EEG headsets have been used for real-time mental state monitoring (Sadeghi et al. 2016). These are then used to mitigate external conditions and stimuli that can affect patients' mental state.

All these applications require low latency and reliable analytics to be performed over observations that are being collected by wearables. Given that such devices need to be lightweight, the computation is often outsourced to a smartphone or a server that acts as a fog resource and to which the observational data is passed for computation. There are also decisions to be made with respect to what to compute on the wearable and what to communicate to the fog, so as to balance the energy usage of the device.

Mobility

Driverless cars and drones are emerging application domains where fog computing plays a key role (Chiang and Zhang 2016). Both these platforms have many onboard sensors and real-time analytics for autonomous mobility. Vehicular networks allow connected cars to communicate with each other (V2V) to cooperatively share information to make decisions on traffic and road conditions (Hou et al. 2016). These can be extended to share compute capacity to perform these analytics. This allows a collection of parked or slow-moving vehicles to form a fog of resources among proximate ones. These may complement occasional roadside units that offer connectivity with the cloud. Here, the entities forming the fog can change dynamically and requires distributed resource coordination.

Drones are finding use in asset monitoring, such as inspecting gas pipelines and power transmission towers at remote locations (Loke 2015). Here, a mobile base station has a digital control tether with a collection of drones that follow a preset path to collect observational data. The

drones have limited compute capacity to increase their endurance and need to prioritize its use for autonomous navigation. So they typically serve as "data mules," collecting data from sensors along the route but doing limited onboard processing (Misra et al. 2015). However, when situations of interest arise, they can make use of their local capacity, resources available with nearby drones, or with fog resources in the base station, while the trip is ongoing. This can help decide if an additional flypast is necessary.

Research Directions

Fog computing is still exploratory in nature. While it has gained recent attention from researchers, many more topics need to be examined further.

Fog Architectures. Many of the proposed fog architectural designs have not seen large-scale deployments and are just plausible proposals. While city-scale deployments with 100s of fog devices are coming online, the experiences from their operations will inform future design (Yannuzzi et al. 2017). Network management is likely to be a key technical challenge as traffic management within the metropolitan area network (MAN) gains importance (Vaquero and Rodero-Merino 2014). Unlike static fog resources, mobile or ad hoc resources such as vehicular fog will pose challenges of resource discovery, access, and coordination (Hou et al. 2016; He et al. In Press). Resource churn will need to be handled through intelligent scheduling (Garcia Lopez et al. 2015). Resources will also require robust adaptation mechanisms based on the situational context (Preden et al. 2015). Open standards will be required to ensure interoperability. To this end, there are initial efforts on defining reference models for fog computing (Byers and Swanson 2017).

Data Management. Tracking and managing content across edge, fog, and cloud will be a key challenge. Part of this is the result of devices in IoT acting as data sources and compute

platforms, which necessitates coordination across the cyber and physical worlds (Anand et al. 2017). The generation of event streams that are transient and need to be processed in a timely manner poses additional challenges to the velocity dimension of Big Data (Naas et al. 2017). Data discovery, replication, placement, and persistence will need careful examination in the context of wide area networks and transient computing resources. Sensing will need to be complemented with “sensemaking” so that data is interpreted correctly by integrating multiple sources (Preden et al. 2015).

Programming Models and Platforms. Despite the growth of edge and fog resources, there is a lack of a common programming abstraction or runtime environments for defining and executing distributed applications on these resources (Stoica et al. 2017). There has been some preliminary work in defining a hierarchical pattern for composing applications that generate data from the edge and need to incrementally aggregate and process them at the fog and cloud layers (Hong et al. 2013). They use spatial partitioning to assign edge devices to fogs. The ECHO platform offers a dataflow model to compose applications that are then scheduled on distributed runtime engines that are present on edge, fog, and cloud resources (Ravindra et al. 2017). It supports diverse execution engines such as NiFi, Storm, and TensorFlow, but the user couples the tasks to a specific engine. A declarative application specification and Big Data platform are necessary to ease the composition of applications in such complex environments.

Related to this are application deployment and resource scheduling. VMs are used in the cloud to configure the required environment, but they may prove to be too resource intensive for fog. Some have examined the use of just a subset of the VM’s footprint on the fog and migrating this image across resources to track the mobility of user(s) who access its services (Bittencourt et al. 2015). Resource scheduling on edge, fog, and cloud has also been explored, though often validated just through simulations due to the lack of access to large-scale fog setups (Gupta et al. 2017; Zeng et al. 2017). Spatial awareness

and energy awareness are distinctive features that have been included in such schedulers (Brogi and Forti 2017; Ghosh and Simmhan In Press). Formal modeling of the fog has been undertaken as well (Sarkar and Misra 2016). Quality of Experience (QoE) as a user-centric alternative metric to Quality of Service (QoS) is also being examined (Aazam et al. 2016).

Such research will need to be revisited as the architectural and application models for fog computing become clearer, with mobility, availability, and energy usage of resources offering unique challenges (Shi et al. 2012).

Security, Privacy, Trust. Unlike cloud computing where there is a degree of trust in the service provider, fog computing may contain resources from diverse and ad hoc providers. Further, fog devices may not be physically secured like a data center and may be accessible by third parties (Chiang and Zhang 2016). Containerization does not offer the same degree of sand-boxing between multiple tenant applications that virtualization does. Hence, data and applications in the fog operate within a mix of trusted and untrusted zones (Garcia Lopez et al. 2015). This requires constant supervision of the device, fabric, applications, and data by multiple stakeholders to ensure that security and privacy are not compromised. Techniques like anomaly detection, intrusion detection, moving target defense, etc. will need to be employed. Credential and identity management will be required. Provenance and auditing mechanisms will prove essential as well. These will need to be considered as first-class features when designing the fog deployment or the application.

Cross-References

- ▶ [Big Data Analysis and IOT](#)
- ▶ [Big Data in Mobile Networks](#)
- ▶ [Big Data in Smart Cities](#)
- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Mobile Big Data: Foundations, State of the Art, and Future Directions](#)

References

- Aazam M, Huh EN (2014) Fog computing and smart gateway based communication for cloud of things. In: International conference on future internet of things and cloud (FiCloud)
- Aazam M, St-Hilaire M, Lung CH, Lambadaris I (2016) Mefore: QoE based resource estimation at fog to enhance QoS in IoT. In: International conference on telecommunications (ICT)
- Anand N, Chintalapally A, Puri C, Tung T (2017) Practical edge analytics: architectural approach and use cases. In: IEEE international conference on edge computing (EDGE)
- Aral A, Brandic I (2017) Quality of service channelling for latency sensitive edge applications. In: IEEE international conference on edge computing (EDGE)
- Amrutmurt B et al (2017) An open smart city IoT test bed: street light poles as smart city Spines. In: ACM/IEEE international conference on internet-of-things design and implementation (IoTDI)
- Barbera MV, Kosta S, Mei A, Stefa J (2013) To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In: IEEE international conference on computer communications (INFOCOM)
- Beck MT, Werner M, Feld S, Schimper T (2014) Mobile edge computing: a taxonomy. In: International conference on advances in future Internet
- Bittencourt LF, Lopes MM, Petri I, Rana OF (2015) Towards virtual machine migration in fog computing. In: IEEE international conference on P2P, parallel, grid, cloud and Internet computing (3PGCIC)
- Bonomi F (2011) Cloud and fog computing: trade-offs and applications. In: Workshop on network-optimized computing at the edge of the network (EON)
- Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: a platform for Internet of things and analytics. In: Bessis N, Dobre C (eds) Big data and Internet of things: a roadmap for smart environments. Springer, Cham, pp 169–186
- Brogi A, Forti S (2017) QoS-aware deployment of IoT applications through the fog. *IEEE Internet Things J* 4(5):1–8
- Byers C, Swanson R (2017) Openfog reference architecture for fog computing. Technical report, Open Fog Consortium
- Cai H, Xu B, Jiang L, Vasilakos AV (2017) IoT-based big data storage systems in cloud computing: perspectives and challenges. *IEEE Internet Things J* 4(1): 75–87
- Cao Y, Chen S, Hou P, Brown D (2015) Fast: a fog computing assisted distributed analytics system to monitor fall for stroke mitigation. In: IEEE international conference on networking, architecture and storage (NAS)
- Chiang M, Zhang T (2016) Fog and IoT: an overview of research opportunities. *IEEE Internet Things J* 3(6):854–864
- Dastjerdi AV, Buyya R (2016) Fog computing: helping the Internet of things realize its potential. *IEEE Comput* 49(8):112–116
- Donnet B, Friedman T (2007) Internet topology discovery: a survey. *IEEE Commun Surv Tutor* 9(4):56–69
- Fernando N, Loke SW, Rahayu W (2013) Mobile cloud computing: a survey. *Futur Gener Comput Syst* 29(1):84–106
- Garcia Lopez P, Montresor A, Epema D, Datta A, Higashino T, Iamnitchi A, Barcellos M, Felber P, Riviere E (2015) Edge-centric computing: vision and challenges. *ACM SIGCOMM Comput Commun Rev* 45(5):37–42
- Ghosh R, Simmhan Y (In Press) Distributed scheduling of event analytics across edge and cloud. *ACM Trans Cyber Phys Syst*
- Gubbi J, Buyya R, Marusic S, Palaniswami M (2013) Internet of things (IoT): a vision, architectural elements, and future directions. *Futur Gener Comput Syst* 29(7):1645–1660
- Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R (2017) iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of things, edge and fog computing environments. *Softw Pract Exp* 47(9):1275–1296
- He K, Fisher A, Wang L, Gember A, Akella A, Ristenpart T (2013) Next stop, the cloud: understanding modern web service deployment in EC2 and azure. In: Conference on Internet measurement
- He J, Wei J, Chen K, Tang Z, Zhou Y, Zhang Y (In Press) Multi-tier fog computing with large-scale IoT data analytics for smart cities. *IEEE Internet Things J*. <https://doi.org/10.1109/JIOT.2017.2724845>
- Hong K, Lillethun D, Ramachandran U, Ottenwälder B, Koldehofe B (2013) Mobile fog: a programming model for large-scale applications on the Internet of things. In: ACM SIGCOMM workshop on mobile cloud computing
- Hou X, Li Y, Chen M, Wu D, Jin D, Chen S (2016) Vehicular fog computing: a viewpoint of vehicles as the infrastructures. *IEEE Trans Veh Technol* 65(6):3860–3873
- Hwang K, Dongarra J, Fox GC (2013) Distributed and cloud computing: from parallel processing to the Internet of things. Morgan Kaufmann, Waltham, MA
- Jalali F, Khodadustan S, Gray C, Hinton K, Suits F (2017) Greening IoT with fog: a survey. In: IEEE international conference on edge computing (EDGE)
- Khochare A, Ravindra P, Reddy SP, Simmhan Y (2017) Distributed video analytics across edge and cloud using echo. In: International conference on service-oriented computing (ICSOC) demo
- Lewis G, Echeverría S, Simanta S, Bradshaw B, Root J (2014) Tactical cloudlets: moving cloud computing to the edge. In: IEEE military communications conference (MILCOM)
- Loke SW (2015) The Internet of flying-things: opportunities and challenges with airborne fog computing and mobile cloud in the clouds. Technical report 1507.04492v1, arXiv
- Milojicic DS, Kalogeraki V, Lukose R, Nagaraja K, Pruyne J, Richard B, Rollins S, Xu Z (2002) Peer-to-peer computing. Technical report HPL-2002-57, HP labs

- Misra P, Simmhan Y, Warrior J (2015) Towards a practical architecture for internet of things: an India-centric view. *IEEE Internet of Things (IoT) Newsletter*
- Naas MI, Parvedy PR, Boukhobza J, Lemarchand L (2017) iFogStor: an IoT data placement strategy for fog infrastructure. In: IEEE international conference on fog and edge computing (ICFEC)
- Nelson DL, Bell CG (1986) The evolution of workstations. *IEEE Circuits and Devices Mag* 2(4): 12–16
- Pradhan S, Dubey A, Khare S, Nannapaneni S, Gokhale A, Mahadevan S, Schmidt DC, Lehofer M (In Press) Chariot: goal-driven orchestration middleware for resilient IoT systems. *ACM Trans Cyber-Phys Syst*
- Preden JS, Tammemae K, Jantsch A, Leier M, Riid A, Calis E (2015) The benefits of self-awareness and attention in fog and mist computing. *IEEE Comput* 48(7):37–45
- Ravindra P, Khochare A, Reddy SP, Sharma S, Varshney P, Simmhan Y (2017) Echo: an adaptive orchestration platform for hybrid dataflows across cloud and edge. In: International conference on service-oriented computing (ICSOC)
- Sadeghi K, Banerjee A, Sohankar J, Gupta SK (2016) Optimization of brain mobile interface applications using IoT. In: IEEE international conference on high performance computing (HiPC)
- Sarkar S, Misra S (2016) Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *IET Netw* 5(2):23–29
- Satyanarayanan M, Bahl P, Caceres R, Davies N (2009) The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Comput* 8(4):1536–1268
- Satyanarayanan M, Simoens P, Xiao Y, Pillai P, Chen Z, Ha K, Hu W, Amos B (2015) Edge analytics in the Internet of things. *IEEE Pervasive Comput* 14(2):1536–1268
- Shi C, Lakafosis V, Ammar MH, Zegura EW (2012) Serendipity: enabling remote computing among intermittently connected mobile devices. In: ACM international symposium on mobile ad hoc networking and computing (MobiHoc)
- Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. *IEEE Internet Things J* 3(5):2327–4662
- Shukla A, Chaturvedi S, Simmhan Y (2017) RIoTBench: an IoT benchmark for distributed stream processing systems. *Concurr Comput: Pract Exp* 29:e4257
- Simmhan Y (2017) IoT analytics across edge and cloud platforms. *IEEE IoT Newsletter*. <https://iot.ieee.org/newsletter/may-2017/iot-analytics-across-edge-and-cloud-platforms.html>
- Simmhan Y, Aman S, Kumbhare A, Liu R, Stevens S, Zhou Q, Prasanna V (2013) Cloud-based software platform for big data analytics in smart grids. *Comput Sci Eng* 15(4):38–47
- Sinha A (1992) Client-server computing. *Commun ACM* 35(7):77–98
- Stoica I, Song D, Popa RA, Patterson DA, Mahoney MW, Katz RH, Joseph AD, Jordan M, Hellerstein JM, Gonzalez J et al (2017) A Berkeley view of systems challenges for AI. Technical report, University of California, Berkeley
- Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. *ACM SIGCOMM Comput Commun Rev* 44(5):27–32
- Varshney P, Simmhan Y (2017) Demystifying fog computing: characterizing architectures, applications and abstractions. In: IEEE international conference on fog and edge computing (ICFEC)
- Yannuzzi M, van Lingen F, Jain A, Parellada OL, Flores MM, Carrera D, Perez JL, Montero D, Chacin P, Corsaro A, Olive A (2017) A new era for cities with fog computing. *IEEE Internet Comput* 21(2):54–67
- Yi S, Li C, Li Q (2015) A survey of fog computing: concepts, applications and issues. In: Workshop on mobile big data
- Zeng X, Garg SK, Strazdins P, Jayaraman PP, Georgakopoulos D, Ranjan R (2017) IOTSim: a simulator for analysing IoT applications. *J Syst Archit* 72:93–107
- Zhou Q, Simmhan Y, Prasanna VK (2017) Knowledge-infused and consistent complex event processing over real-time and persistent streams. *Futur Gener Comput Syst* 76:391–406

Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems

Adnan Mahmood^{1,2,3}, Hushairi Zen², and Shadi M. S. Hilles³

¹Telecommunication Software and Systems Group, Waterford Institute of Technology, Waterford, Ireland

²Universiti Malaysia Sarawak, Kota Samarahan, Sarawak, Malaysia

³Al-Madinah International University, Shah Alam, Selangor, Malaysia

Definitions

Intelligent Transportation System is regarded as a smart application encompassing the promising features of sensing, analysis, control, and communication technologies in order to improve

the safety, reliability, mobility, and efficiency of ground transportation.

Big Data is an evolving and emerging terminology describing a voluminous amount (*terabytes, petabytes, exabytes, or even zettabyte*) of structured, unstructured, and semi-structured data which could be mined for information.

Overview

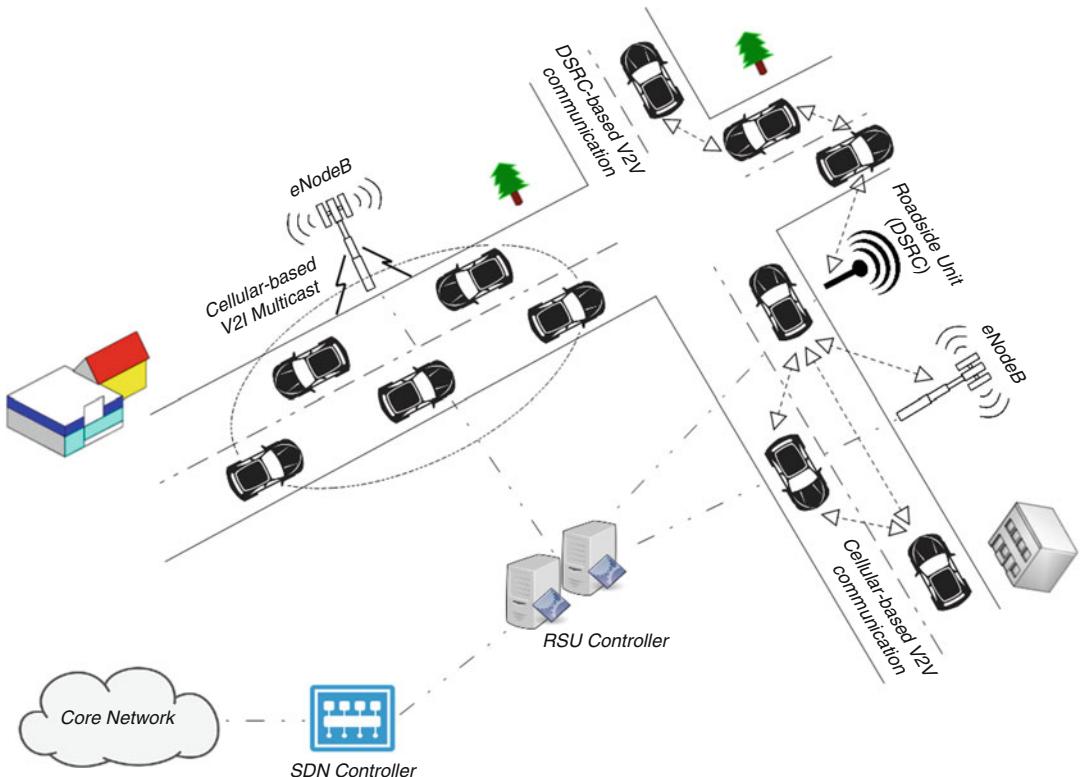
The evolution of Big Data in large-scale Internet-of-Vehicles has brought forward unprecedented opportunities for a unified management of the transportation sector, and for devising smart Intelligent Transportation Systems. Nevertheless, such form of frequent heterogeneous data collection between the vehicles and numerous applications platforms via diverse radio access technologies has led to a number of security and privacy attacks, and accordingly demands for a ‘secure data collection’ in such architectures. In this respect, this chapter is primarily an effort to highlight the said challenge to the readers, and to subsequently propose some security requirements and a basic system model for secure Big Data collection in Internet-of-Vehicles. Open research challenges and future directions have also been deliberated.

Introduction

Over the past few decades, a rapid increase in the number of vehicles (both passenger and commercial) across the globe has pushed the present-day transportation sector to its limits. This has thus caused the transportation systems to become highly ineffective and relatively expensive to maintain and upgrade overtime (Contreras et al. 2017). According to one recent estimate (British Petroleum 2017), the number of vehicles on the road has already surpassed 0.9 billion and is likely to be doubled by the end of year 2035. This massive increase in the number of vehicles has not only led to extreme traffic congestion(s) in dense urban environments and hinders economic growth in several ways but also transpires

in a number of road fatalities. The World Health Organization (2017) estimates that around 1.25 million people across the world die each year and millions more get injured as a result of road accidents, with nearly half of them being vulnerable road users, i.e., pedestrians, roller skaters, and motorcyclists. Approximately 90% of these road accidents occurs in low- and middle-income economies as a result of their poor infrastructures. From a network management perspective, this results in serious deterioration of quality of service for numerous safety-critical and non-safety (i.e., infotainment) applications and in degradation of quality of experience of vehicular users. Thus, there exists a significant room for improvement in the existing transportation systems in terms of enhancing safety, non-safety (i.e., infotainment), and efficiency aspects; and if properly addressed, this would ultimately set tone for highly efficacious intelligent transportation systems indispensable for realizing the vision of connected vehicles.

The paradigm of vehicular communication has been studied and thoroughly analyzed over the past two decades (Ahmed et al. 2017). It has rather evolved from traditional vehicle-to-infrastructure (V2I) communication to more recent vehicle-to-vehicle (V2V) communication and vehicle-to-pedestrian (V2P) communication, thus laying the foundations for the futuristic notion of vehicle-to-everything (V2X) communication (Seo et al. 2016). While cellular networks have been mostly relied upon for V2V communication due to its extended communication range (i.e., theoretically up to 100 km) and extremely high data rates, its high cost and low reliability in terms of guaranteeing stringent delay requirements do not make it as an ideal mode of communication in highly dynamic networks. On the contrary, the evolution of dedicated short-range communication (DSRC) as a two-way short-to-medium range wireless communication for safety-based V2V applications has been a subject of attention in recent years for engineers and scientists as a result of its efficacious real-time information exchange between the vehicles. While a number of challenges still hinder the effective deployment



Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems, Fig. 1
Heterogeneous vehicular networking architecture

of DSRC-based vehicular networks, the US Department of Transportation (US DoT) is actively pursuing the same as one of its major research priority for various public safety and traffic management applications including, but not limited to, forward collision warnings, blind intersection collision mitigation, (approaching) emergency vehicle warnings, lane change assistance, and for anticipated traffic and travel conditions (Lu et al. 2014). A topological diagram of next-generation heterogeneous vehicular networks is depicted in Fig. 1.

In addition to both cellular networks and DSRC, numerous other radio access technologies (RATs) are currently being explored for vehicular communication. Since the number of sensors deployed onboard the vehicles (and on roadside infrastructure) is anticipated to increase by manifold primarily due to the introduction of connected and autonomous vehicles, it is neces-

sary to deploy a wireless communication system with the ability to transmit a large amount of data at high data rates. Since DSRC typically allows vehicles to transmit data up to a range of 1 km and with a practical data rate of 2–6 Mbps, and while data rates for cellular networks are also limited to 100 Mbps in high mobility scenarios, the ITS community has also gained momentum for exploring the possibility of millimeter wave communication (mmWave) for vehicular networking. mmWave is expected to offer gigabit-per-second data rate; however, issues such as efficacious channel modeling, security, and beam alignment still pose a considerable challenge to its implementation (Va et al. 2016). Further higher-frequency bands in the frequency spectrum are also gaining attention, and terahertz communication for vehicular networks is also on the cards of researchers in academia and wireless industry, thus paving the way for fifth-generation (5G) and

beyond fifth-generation (beyond 5G) wireless networking technologies (Mumtaz et al. 2017).

However, none of these aforementioned technologies possess a potential to fully realize the breadth of vehicular safety and non-safety applications simultaneously and especially when their requirements are in conflict with one another. The ideal approach thus is to ensure a synergy of several RATs so as to provide an appropriate heterogeneous vehicular networking platform in order to meet the stringent communication requirements. Heterogeneity is an important and timely topic but beyond the scope of this work. Since a vehicular network encompasses numerous heterogeneous data sources (i.e., not only from heterogeneous RATs but also from diverse sort of sensing equipment), the amount of data generated is not only huge but also poses a significant challenge to the network's security and stability. Therefore, this chapter, in contrast to other commonly published surveys and chapters, primarily focuses on secure Big Data collection in vehicular networks (interchangeably, referred to as *Internet of Vehicles* in this chapter).

Toward Big Data Collection in Vehicular Networks

In the near future, several substantial changes are anticipated in the transportation industry especially in light of the emerging paradigms of Internet of Things (IoTs), cloud computing, edge and/or fog computing, software-defined networking, and more recently Named Data Networking, among many others. Such promising notions have undoubtedly strengthened and contributed to the industrial efforts of devising intelligently connected vehicles for developing safe and convenient road environments. Furthermore, with the rapid evolution of high-speed mobile Internet access and the demand for a seamless ubiquitous communication at a much affordable price, new vehicular applications/services are emerging. The European Commission estimates that around 50–100 billion smart devices would be directly connected to the Internet by the end of year 2019, and approximately 507.5 ZB/year of data would be

produced due to the same (Sun and Ansari 2016). Moreover, today the smart vehicles have emerged as a multisensor platform, as the usual number of intelligent sensors installed on a vehicle is typically around 100, and this number is anticipated to increase to 200 by year 2020 (Choi et al. 2016). The data streams generated via these sensors are not only high in volume but also possess a fast velocity due to highly dynamic nature of vehicular networks. Moreover, information generated from numerous social networking platforms (i.e., Weibo, Facebook, Twitter, WhatsApp, WeChat) are regularly accessed by the vehicular users and generate a lot of traffic data in real time. This data is spatiotemporal in nature due to its dependence on time and geographical location of vehicles. Also, since the trajectory of vehicles is usually reliant on the road distribution across a huge geographical region, the collected information is also heterogeneous, multimodal, and multidimensional and varies in its quality. This context is pushing the concept of traditional vehicular ad hoc networks to large-scale Internet of Vehicles (IoVs), and all of this collected data converges as Big Data in vehicular networks which ultimately is passed via the core networks to both regional and centralized clouds.

Nevertheless, today's Internet architecture is not yet scalable and is quite inefficient to handle such a massive amount of IoV Big Data. Also, transferring of such data is relatively time-consuming and expensive and requires a massive amount of bandwidth and energy. Moreover, real-time processing of this collected information is indispensable for a number of safety-critical applications and hence demands for a highly efficient data processing architecture with a lot of computational power. As vehicular networks are highly distributive in nature, a distributed edge-based processing is recommended over traditional centralized architectures. However, such distributed architectures mostly have limitations of compute and storage and therefore are unable to cache and process a huge amount of information. Last but not the least, it is quite significant to design a secure mechanism which ensures that collection of IoV Big Data is trusted and not tampered with. There is a huge risk of fraudulent

messages injected by a malicious vehicle that could easily endanger the whole traffic system(s) or could potentially employ the entire network to pursue any dangerous activity for its own wicked benefits. Thus, *how to effectively secure the Big Data collection in IoVs deserves researching*. In this context, this chapter is of high importance so as to bring to forth the significance of such challenges for the attention of academic and industrial research community.

Security Requirements and System Model for Secure Big Data Collection in IoVs

Big Data primarily has three main characteristics, (a) volume, (b) variety, and (c) velocity, and this is also referred to as 3Vs (Guo et al. 2017). Vehicles usually collect a massive amount of data from diverse geographical areas and with various heterogeneous attributes, thus ultimately converging to IoV Big Data with variation in its size, volume, and dimensionality. The analytics of such Big Data can help network operators to optimize the overall resource(s) planning of next-generation vehicular networks. It would also facilitate the national transportation agencies to critically analyze and subsequently mitigate the dense traffic problems in an efficient manner, in turn making the lives of millions of people comfortable and convenient. These are some of the potential advantages that have lately encouraged the vehicles' manufacturers to build large-scale Big Data platforms for the intelligent transportation systems. However, all of this data analytics and subsequent decision making becomes impractical if any malicious vehicle is able to inject data in the IoV Big Data stream which would have severe implications for both safety and non-safety vehicular applications. In terms of safety applications, any maliciously fed information may lead to false trajectory predictions and wrong estimation of a vehicle's neighboring information in the near future which could prove quite fatal for passengers traveling in both semi-autonomous and fully autonomous vehicles. In terms of non-safety applications, this could not

only lead to a considerable delay in the requested services but also exposes a user's privacy data to extreme vulnerabilities (Singh et al. 2016; Zheng et al. 2015). Thus, any secure IoV information collection scheme should adhere to the following requirements so as to guarantee the secure Big Data collection:

Data Authentication – to verify the identities of the vehicles or vehicular clouds (i.e., vehicles with similar objectives and in vicinity of one another form vehicular clouds for resource-sharing purposes)

Data Integrity – to ensure that the transmitted data has been delivered correctly without any modification or destruction, as this would ultimately become part of the Big Data stream

Applications/Services Access Control – to warrant that each respective vehicle has access to the applications/services it is only entitled for

Data Confidentiality – to guarantee a secure communication among vehicles participating in a networking environment

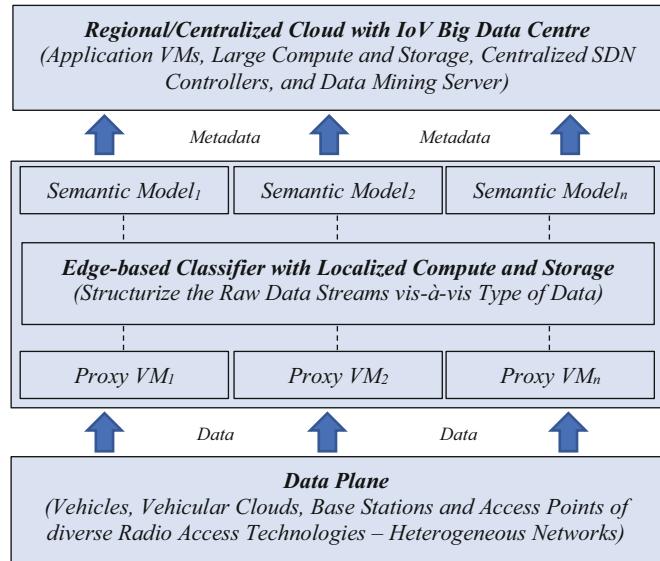
Data Non-repudiation – to ensure that any particular vehicle could not deny the authenticity of another vehicle

Anti-jamming – to prevent intrusion attempts from malicious vehicles which may partially or completely choke the entire network

An ideal approach to tackle these requirements would be via a hierarchical edge-computing architecture as depicted in Fig. 2. Each vehicle (or a vehicular cluster), along with their respective vehicular users, is associated with their private proxy VMs situated in the edge node which not only collects the (raw) data streams from their registered network entities via their respective base stations or access points but also classifies them into various groups depending on the type of data and subsequently generates metadata which is passed to regional/centralized IoV Big Data centers. The metadata encompasses certain valuable information, i.e., geographical locations of vehicles and corresponding timestamps, type of vehicular applications accessed or infotainment services and contents requested, and a pattern recognition identifier that checks for any irregularities in data stream against various

Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems,

Fig. 2 Edge computing architecture for secure IoV
Big Data collection



prescribed network operator policies and also by comparing it with previously known patterns. In this way, a suspicious content could be filtered out at the edge without passing it to the regional/centralized storage. The metadata only contains valuable information without violating a user's privacy. Also, as several vehicles and vehicular users requests similar type of applications/services and pass identical information about their surroundings (i.e., provided if they are in vicinity of one another), intelligent content similarity schemes should be employed in order to avoid any duplication that may lead to excessive computational overheads both at the edge and at the regional/centralized cloud level.

Thus, in this way, not only the computational overheads can be considerably reduced but the amount of IoV Big Data that needs to be cached can also be significantly minimized. It is therefore pertinent to mention that (both) instantaneous and historical data is of critical importance in making intelligent decision making in IoV architectures. Nevertheless, it is also not practically possible to store all of the historical IoV Big Data for an infinite time due to storage issues globally. Thus, it is of extreme importance to not only secure the collection of IoV Big Data but to also ensure that the said data streams should not be excessively cached but disposed after a

certain time. Our envisaged architecture hence proposes to structure the raw data streams into valuable metadata, which not only consumes less resources in making a requisite decision but also takes less storage space in contrast to raw data streams.

Open Research Challenges and Future Directions

Unlike traditional IoT entities which are generally static in nature, IoV is a highly dynamic organism, and this feature makes it quite difficult to effectively deploy the next-generation intelligent transportation system platforms, at least in their current form. Since vehicles usually traverses at quite high geographical speeds, their neighborhood also changes dynamically. Thus, it is challenging to accurately apprehend a vehicle's current and anticipated surrounding neighborhood on run-time basis. A major problem is that by the time the IoV information has been captured, analyzed, and presented to network operators or interested vehicles/vehicular users, the real-time situation has already changed, and the IoV data recently presented has become obsolete. Thus, IoV standalone might not make any essence in the long run; rather, it has to be further optimized with an emerging yet promising paradigm of

software-defined networking which through its centralized controllers (possessing a globalized topological view) addresses the trajectory and neighborhood estimation problems to a much greater extent.

The other issue pertains to density of the vehicular networks. If each individual vehicle starts communicating with its edge (i.e., for reporting status information or requesting vehicular applications or infotainment services), this could transpire in a network broadcast storm that may severely deteriorate the overall network performance, and consequently the service-level objectives may not be fulfilled. In this respect, a notion of vehicular cloud has been proposed in order to mitigate the said threat. However, for this purpose, efficacious *vehicular cloud formation schemes* have to be envisaged. Although the concept of vehicular cloud is much similar to that of a cluster formation in sensor networks or more precisely to vehicular platoons in vehicular ad hoc networks, the actual challenge here is again the dynamic nature of vehicular networks due to which vehicular clouds could make and break at a rapid pace resulting in overall wastage of network resources. Also, it is important that no malicious vehicle becomes part of the vehicular cloud, as this could easily become fatal for the entire IoV environment.

Finally, with a recent push for openness of IoV data sharing among diverse cloud service providers, it has become quite imminent that each cloud service provider should protect its infrastructures from a diverse range of malicious attacks by employing specialized access controls, isolation, encryption, and sanitization techniques. Hence, the input data from other cloud service providers need to pass via a validation process so that any malicious, faulty, or compromised data may be timely detected and subsequently blocked in order to protect the data integrity of the whole IoV ecosystem.

Cross-References

- ▶ [Big Data Analysis for Smart City Applications](#)
- ▶ [Big Data in Smart Cities](#)

Acknowledgments

The corresponding author would like to acknowledge the generous support of the Ministry of Higher Education, Government of Malaysia, for supporting the said research work through its Malaysian International Scholarship Grant, KPT. 600-4/1/12 JLID 2(8).

References

- Ahmed SH et al (2017) Named data networking for software defined vehicular networks. *IEEE Commun Mag* 55(8):60–66. <https://doi.org/10.1109/MCOM.2017.1601137>
- Choi J et al (2016) Millimeter wave vehicular communication to support massive automotive sensing. *IEEE Commun Mag* 54(12):160–167. Available at: <http://arxiv.org/abs/1602.06456>
- Contreras J, Zeadally S, Guerrero-Ibanez JA (2017) Internet of vehicles: architecture, protocols, and security. *IEEE Internet Things J* 99:1–9. <https://doi.org/10.1109/JIOT.2017.2690902>
- Guo L et al (2017) A secure mechanism for big data collection in large scale internet of vehicle. *IEEE Internet Things J* 4(2):601–610. <https://doi.org/10.1109/JIOT.2017.2686451>
- Lu N et al (2014) Connected vehicles: solutions and challenges. *IEEE Internet Things J* 1(4):289–299. <https://doi.org/10.1109/JIOT.2014.2327587>
- Mumtaz S et al (2017) Terahertz communication for vehicular networks. *IEEE Trans Veh Technol* 66(7):5617–5625. <https://doi.org/10.1109/TVT.2017.2712878>
- Organization WH (2017) Save lives – a road safety technical package. Available at: <http://apps.who.int/iris/bitstream/10665/255199/1/9789241511704-eng.pdf?ua=1>
- Petroleum B (2017) BP energy outlook, *British petroleum*. Available at: <https://www.bp.com/content/dam/bp/pdf/energy-economics/energy-outlook-2017/bp-energy-outlook-2017.pdf>
- Seo H et al (2016) LTE evolution for vehicle-to-everything services. *IEEE Commun Mag* 54(6):22–28. <https://doi.org/10.1109/MCOM.2016.7497762>
- Singh J et al (2016) Twenty security considerations for cloud-supported internet of things. *IEEE Internet Things J* 3(3):269–284. <https://doi.org/10.1109/JIOT.2015.2460333>
- Sun X, Ansari NE (2016) Edge IoT: mobile edge computing for the Internet of things. *IEEE Commun Mag* 54(12):22–29. <https://doi.org/10.1109/MCOM.2016.1600492CM>
- Va V et al (2016) Millimeter wave vehicular communications: a survey. *Found Trends Netw* 10(1):1–113. <https://doi.org/10.1561/1300000054>

Zheng X et al (2015) Big data for social transportation. *IEEE Trans Intell Transp Syst* 17(3):620–630. <https://doi.org/10.1109/TITS.2015.2480157>

B

Big Data and Recommendation

Giancarlo Sperlì

Department of Electrical Engineering and Information Technology, University of Naples “Federico II”, Naples, Italy

Synonyms

Data management; Information filtering; Knowledge discovery

Definitions

Recommender systems have been introduced to facilitate the browsing of items' collection assisting users to find “what they need” within this ocean of information that match their preferences or needs.

Overview

This chapter firstly provides an overview of traditional recommender systems and successively it analyzes how the large amount of data, namely “Big Data”, produced by heterogeneous sources changed the traditional approaches with particular attention on the On-line social networks (OSNs) context.

Introduction

The increase of data produced by heterogeneous data sources, due to the large amount of connected devices, led to the definition of the big

data concept. Big data is a field concerning the storage, management, and analysis of large and heterogeneous amount of data which require the development of new methodologies and technologies for their management and processing. Thus, as defined by “The McKinsey Global Institute” Manyika and Chui (2011) “*Big data refers to data sets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze.*”

In the last years, people are able to produce and share digital data (such as text, audio, images, and video) at an unprecedented rate due to the continuously progress in consumer electronics and to the widespread availability of Internet access. For facilitate browsing of large multimedia repositories and for providing useful recommendation, several algorithms and tools have been proposed. Such tools, usually referred to as recommender systems, collect and analyze usage data in order to identify user interests and preferences that allows to suggest items that meet users' needs.

Recommender Systems

Recommender systems have been introduced to facilitate the browsing of items collections, assisting users to find “*what the need*” within this ocean of information. On one hand, users want to have personalized results, and, on other hand, they are not willing to spend too much time to specify their personal information. In the mid-1990s, early works of Resnick et al. (1994) about recommender systems have been proposed to address the problem of information overloaded with the aim to provide useful suggestions based on users' preferences about items that could be of interest (Ricci et al. 2011a).

Recommendation algorithms have been used in several fields, such as *e-commerce Web sites*, in which information about a customer's interests are used to generate a list of recommended items. Other applications exploit the features of item that customers purchase, the feedback on the quality of the product, and the seller trustworthiness and explicitly rate to provide useful sug-

gestions. Moreover, they can use other attributes, such as viewed items, demographic data, subject interests, and favorite artists. *Recommender systems* automatically identify relevant information for a given user, learning from available data (user actions, user profiles). In particular, a *recommender system* supports users to identify the most interesting items, helping people confused by multiple choices analyzing the behavior of the people who have similar interests to them.

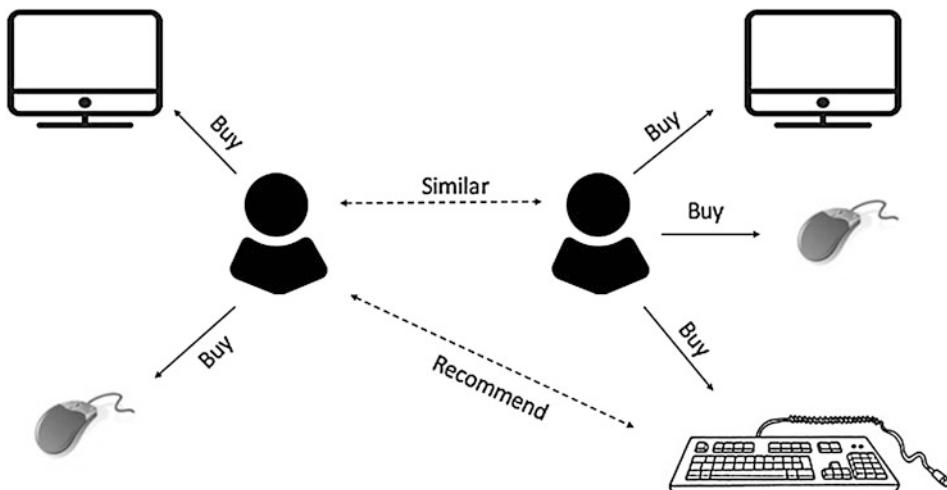
In a traditional model of *recommender system*, there are two types of entities: “*user*” and “*item*”. Users (often customers) have associated metadata (or content) such as age, gender, race, and other demographic information. Items (often products) also have metadata such as text description, price, weight, etc. Different interactions can be made between these two types of entity: user *A* downloads/purchases movie *B*, and user *X* gives a rating 5 to the product *Y* and so on.

The last-generation architecture of recommender systems is composed by one or more of the following stages (Ricci et al. 2011b; Colace et al. 2015): when a given user *i* is browsing a particular items’ collection, the system initially determines a set of useful *candidate* items $O_i^c \subset O$ based on user preferences and needs (*pre-filtering stage*); successively it assigns *score* (or a *rank*) $r_{i,j}$ to each item o_j in O_i^c , computed by analyzing the characteristics of each item

(feedbacks, past behaviors, and so on) (*ranking stage*); finally, it selects a subset of the item best suited (*post-filtering stage*).

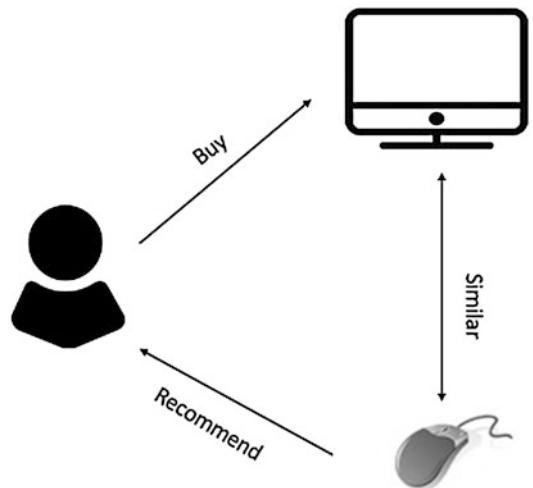
The recommender system approaches are usually classified into the following three categories:

- *Collaborative Filtering*: these systems rely solely on user’s preferences to make recommendations. Until a new item is rated by an appropriate number of users, the recommender system would not be able to provide meaningful recommendations. This problem is known as *cold-start* problem due to an initial lack of ratings. This approach is divided into (Fig. 1):
 - *Memory-based* approaches identify interesting items from other similar users’ opinions by performing the nearest neighbors on a rating matrix. This heuristics make rating predictions based on the entire collection of previously rated items by the users.
 - *Model-based* approaches use the collection of ratings to learn a model, which is then used to make ratings’ prediction.
- *Content-based filtering* approaches provide their suggestion based on the content of items, described as a set of tags, and the user profiles. The user may be recommended items similar



Big Data and Recommendation, Fig. 1 Collaborative filtering

**Big Data and
Recommendation, Fig. 2**
Content-based filtering



- to the ones the user preferred in the past. These approaches have some limitations; indeed the tags are explicitly associated with the items that the systems recommend. Another limitation is that two different items are indistinguishable if they are represented by the same tags. Furthermore, if there is a new user with few ratings, it would not be able to produce accurate recommendations (Fig. 2).
- *Hybrid-filtering based* approach combines both *content-based filtering* and *collaborative filtering* approaches to produce a recommendation. It is possible to classify the hybrid recommender system into the following four categories: (i) approaches that combine the prediction of collaborative and content-based methods; (ii) approaches that integrate some features of content-based into a collaborative approach; (iii) approaches that include some collaborative characteristics into a content-based approach; and (iv) approaches that integrate main characteristics of both models.

The technology behind *recommender systems* has evolved over the past 20 years into a rich collection of tools that enable researchers to develop effective recommender algorithms. Nowadays, the online shopping retails (such as *Amazon*, *Netflix*, etc.) are examples of recommender systems that analyze historical behaviors representing user information related

to past actions to propose additional products or services that enhance the customer satisfaction.

An approach that models recommendations as a social choice is proposed by Albanese et al. (2013). In particular, the authors combine intrinsic features of multimedia, past behaviors of users, and overall behavior of entire community of the users. Moreover, in Zhao et al. (2016), the authors propose a social recommendation framework based on an unified preferences learning process that integrates item content features with collaborative user-item relationships. Finally, Bartolini et al. (2016) propose a recommendation framework that manages heterogeneous data sources to provide context-aware recommendation techniques supporting intelligent multimedia services for the users.

Recommender Systems and Online Social Networks

The development of online social networks (OSNs) led to increase the amount of information on the Web. Different approaches have been proposed to include the information (i.e., users' relationships, behaviors, etc.) of OSNs in the recommender system for properly handling large amount of multimedia contents showing big data features, mainly due to their high change rate, huge volume, and intrinsic heterogeneity.

Big data in social networks (big social data) have been used to customize recommender systems in order to provide appropriate suggestions to users according to their needs and preferences. Also many e-commerce sites use social recommendation system to infer similarity between user purchase behaviors to recommend products that can improve the recommendation process in social networks. It is based on social network information, refers to trust and friendship among users (social relationship that resulted from common interests among users). In particular it is necessary to consider multi-attribute of products and users into the recommendation model. Furthermore, it allows to consider new different features for computing the similarity between two users. These features have three main properties:

1. *Contents generated by users* refer to all contents that are created by the users themselves and their meta-information.
2. *Relationship information* constitutes a social circle representing directly linked or connected relationships between users.
3. *Interaction information* refers to messages or contents exchanged between users.

Leveraging the information contained in OSNs, Qian et al. (2014) propose an unified personalized recommendation model based on probabilistic matrix factorization that combines three social factors: personal interest, interpersonal interest similarity, and interpersonal influence. The last two factors can enhance the intrinsic link among features in the latent space to deal with cold-start problem. In Sun et al. (2015), the authors propose recommender systems based on social networks, called *RSboSN*, which integrates social network graph and the user-item matrix to improve the prediction accuracy of the traditional recommender systems. In particular, a clustering algorithm for the *user-item-tag* matrix is used to identify the most suitable friends for realistic recommendation tasks. Moreover, Colace et al. (2015) propose a collaborative user-centered recommendation approach that combines preferences (usually in the shape of

items metadata), opinions (textual comments to which it is possible to associate a sentiment), behavior (in the majority of cases logs of past items observations made by users), and feedbacks (usually expressed in the form of ratings) with items features and context information for supporting different applications using proper customizations (e.g., recommendation of news, photos, movies, travels, etc.). In Amato et al. (2017) it is proposed a *recommending system* for big data applications based on a collaborative and user-centered approach. The recommendations are made based on the interactions among users and generated multimedia contents in one or more social networks.

In the last years, different recommendation techniques have been proposed in the literature for supporting several kinds of applications in *multimedia social networks*, in which nodes could be users or multimedia objects (photos, video, posts, and so on), and links between nodes could be assumed different meaning (publishing, endorsement, and so on). These systems exploit information extracted by these networks to improve the accuracy of recommendation, and they provide new types of suggestions; for instance, they can recommend not only items but also groups, friends, events, tags, etc. to users through algorithms that it is possible to classify into the following three different classes.

The first one is composed by a set of algorithms leveraging graph as data model in which nodes and edges represent, respectively, single social entities and the relationships among them. Jiang et al. (2015b) propose a modified version of *random walk* on a star-structured heterogeneous network, a particular network centered on social domain which is connected with other domains (i.e., video, Web posts, etc.). The aim of this approach is to predict user-item links in a target domain leveraging the items in auxiliary domains.

Algorithms that try to learn user and item preferences, through factorization of the related rating matrices, compose the second class. Jamali and Lakshmanan (2013) propose *Heteromf*, a context-dependent matrix factorization model that considers jointly two types of latent

factors: general latent factors for each entity type and context-dependent latent factors for each context in which entities are involved.

The third class is constituted by algorithms that model the interactions among users and items as “latent topics,” improving the recommendation process with other types of information. To such purposes, Yuan et al. (2013) propose a probabilistic model to exploit the different types of information (such as spatial, temporal, and activity data) extracted from social networks to infer behavioral models for users. In turn, Pham et al. (2016) propose an algorithm able to measure the influence among different types of entities, with the final aim to achieve higher recommendation accuracy and to understand the role of each entity in recommendation process.

Recommender Systems and Location-Based Social Networks

In addition to the described approaches, the rapid growth of cities led to develop recommendation systems based on points of interest (*POIs*). People, in everyday life, decide *where to go* according to their personal interests. Thus, *POI recommendation* helps users to filter out uninteresting places and reduce their decision-making time. Furthermore, the continuous increase in the use of smartphones led to the emergence of a new type of social network called location-based social networks (LBSNs); the main characteristic of these networks is to integrate the user’s location into the well-known OSN capabilities. With the development of *location-based social networks* (LBSNs) (i.e., FourSquare, Gowalla, etc.), *POI* recommendation on LBSNs (Gao et al. 2015) has recently attracted much attention as the latter provides an unprecedented opportunity to study human mobile behaviors for providing useful recommendations in spatial, temporal, social, and content aspects. These systems allow users to make *check in* at specific POIs with mobile devices and leave tips or share their experience with other users. These facets result in a W^4 (*who, when, where, what*) information layout, corresponding to four distinct information layers.

Content information on LBSNs could be related to a user check-in action; in particular, there are three types of content information:

B

- POI properties related to an action of check in, for example, by observing a POIs description as “vegetarian restaurant”; it is possible to infer that the restaurant serves “vegetarian food” and users who check in at this business object might be interested in the vegetarian diet.
- User interests related to user comments about a particular POI.
- Sentiment indications also related to users comments on a given POI.

The technological development led to see online social network an important means on which make business, allowing firms to promote their products and services and users to provide feedbacks about their experiences. In Zhang et al. (2016), the authors propose an approach based on matrix factorization that exploits both the attributes of items and social links of users to address the cold-start problem for new users by utilizing the social links between users. In particular, Kernel-based attribute-aware matrix factorization (KAMF) method is developed to discover the nonlinear interactions among attributes, users, and items. A recommender system that integrates community-contributed photos, social, and location aspects is proposed by Jiang et al. (2016) for providing a personalized travel sequence recommendation. In this system, similarities between user and route package are used to rank famous routes, whose top one are optimized by analyzing social similar users’ travel records. Moreover, Amato et al. (2014) propose a touristic context-aware recommendation system, based on the experience of previous users and on personal preferences of tourists, which combines information gathered by several heterogeneous sources. Eventually, an author topic model-based collaborative filtering (ATCF) method, based on user preference topics (i.e., cultural, cityscape, or landmark) extracted from the geo-tag constrained textual description of photos, is proposed by Jiang et al. (2015a) to provide *POI* recommendation for social users.

Recommender Systems and Event-Based Social Networks

Nowadays, the Web has grown into one of the most important channels to communicate social events, leading to the rise of a new type of OSN, called event-based social networks (ESBNs) (i.e., Meetup, Plancast, etc.), in which users join in social groups to attend events. Event-based social networks (EBSNs) are online social networks where users can create, promote, and share upcoming events of any kind with other users. The large amount of events available in EBSNs confuses users' ability to choose the events that best fit their interests. In Macedo et al. (2015), the authors propose a context-aware approach to event recommendation that exploits the following different signals that have a positive influence on the user's decision about attending an event: social signals based on group membership, location signals based on the user home distance to each event, and temporal signals derived from the users time preferences. The event participation decisions are analyzed by Ding et al. (2016) considering the interactions on the well-known ESBN Meetup. Successively the authors propose a technique that combines user features from multiple channels to recommend users to new events. In Gao et al. (2016), *SogBmf*, a Bayesian latent factor model that jointly considers social group influence and individual preferences, is proposed for improving the event recommendation performance.

References

- Albanese M, d'Acierno A, Moscato V, Persia F, Picariello A (2013) A multimedia recommender system. *ACM Trans Internet Technol (TOIT)* 13(1):3
- Amato F, Mazzeo A, Moscato V, Picariello A (2014) Exploiting cloud technologies and context information for recommending touristic paths. Springer International Publishing, Cham, pp 281–287. https://doi.org/10.1007/978-3-319-01571-2_33
- Amato F, Moscato V, Picariello A, Sperlí G (2017) Recommendation in social media networks. In: IEEE third international conference on multimedia big data (BigMM) IEEE, pp 213–216
- Bartolini I, Moscato V, Pensa RG, Penta A, Picariello A, Sansone C, Sapino ML (2016) Recommending multi-media visiting paths in cultural heritage applications. *Multimed Tools Appl* 75(7):3813–3842
- Colace F, De Santo M, Greco L, Moscato V, Picariello A (2015) A collaborative user-centered framework for recommending items in online social networks. *Comput Hum Behav* 51:694–704
- Ding H, Yu C, Li G, Liu Y (2016) Event participation recommendation in event-based social networks. In: International conference on social informatics. Springer, pp 361–375
- Gao H, Tang J, Hu X, Liu H (2015) Content-aware point of interest recommendation on location-based social networks. In: AAAI, pp 1721–1727
- Gao L, Wu J, Qiao Z, Zhou C, Yang H, Hu Y (2016) Collaborative social group influence for event recommendation. In: Proceedings of the 25th ACM international on conference on information and knowledge management. ACM, pp 1941–1944
- Jamali M, Lakshmanan L (2013) Heteromf: recommendation in heterogeneous information networks using context dependent factor models. In: Proceedings of the 22nd international conference on World Wide Web. ACM, pp 643–654
- Jiang S, Qian X, Shen J, Fu Y, Mei T (2015a) Author topic model-based collaborative filtering for personalized poi recommendations. *IEEE Trans Multimed* 17(6): 907–918
- Jiang M, Cui P, Chen X, Wang F, Zhu W, Yang S (2015b) Social recommendation with cross-domain transferable knowledge. *IEEE Trans Knowl Data Eng* 27(11): 3084–3097
- Jiang S, Qian X, Mei T, Fu Y (2016) Personalized travel sequence recommendation on multi-source big social media. *IEEE Trans Big Data* 2(1):43–56
- Macedo AQ, Marinho LB, Santos RL (2015) Context-aware event recommendation in event-based social networks. In: Proceedings of the 9th ACM conference on recommender systems. ACM, pp 123–130
- Manyika J, Chui M (2011) Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute, New York
- Pham TAN, Li X, Cong G, Zhang Z (2016) A general recommendation model for heterogeneous networks. *IEEE Trans Knowl Data Eng* 12:3140–3153
- Qian X, Feng H, Zhao G, Mei T (2014) Personalized recommendation combining user interest and social circle. *IEEE Trans Knowl Data Eng* 26(7):1763–1777
- Resnick P, Iacovou N, Suchak M, Bergstrom P, Riedl J (1994) Grouplens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on computer supported cooperative work. ACM, pp 175–186
- Ricci F, Rokach L, Shapira B (2011a) Introduction to recommender systems handbook. In: Recommender systems handbook. Springer, Boston, pp 1–35
- Ricci F, Rokach L, Shapira B, Kantor PB (eds) (2011b) Recommender systems handbook. Springer, Boston

- Sun Z, Han L, Huang W, Wang X, Zeng X, Wang M, Yan H (2015) Recommender systems based on social networks. *J Syst Softw* 99:109–119
- Yuan Q, Cong G, Ma Z, Sun A, Thalmann NM (2013) Who, where, when and what: discover spatio-temporal topics for Twitter users. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 605–613
- Zhang JD, Chow CY, Xu J (2016) Enabling kernel-based attribute-aware matrix factorization for rating prediction. *IEEE Trans Knowl Data Eng* 29(4):798–812
- Zhao Z, Lu H, Cai D, He X, Zhuang Y (2016) User preference learning for online social recommendation. *IEEE Trans Knowl Data Eng* 28(9):2522–2534

Big Data Application in Manufacturing Industry

Amit Mitra¹ and Kamran Munir²

¹Bristol Business School, University of the West of England, Bristol, UK

²Computer Science and Creative Technologies, University of the West of England, Bristol, UK

Introduction

Today data on customers or clients of companies is one of the key resources that adds to the valuation of an organization. Often data is not only generated by customers who are buying from organizations but increasingly because of feedback that they may share on various social media platforms about their experiences of buying. So, traditional approaches where data collection was mainly based on conducting surveys of customer experiences are increasingly becoming a thing of the past. On the one hand, people prefer to refrain from participating in surveys as these may be time-consuming; on the other the amount of time spent on social media platforms has been on the rise. Given such an environment, it is commonplace for organizations to collect data through touch points that customers use when ordering or choosing products and services. Therefore, data generation is not limited to responses to survey questions but more because of touch points and social media posts. In this ever-growing and dynamic need to deal with variety of

formats far exceeds the capacities of traditional modelling approaches. Traditional modelling approaches tend to suit predictive analysis more than those in contexts where a great deal of data is being generated at every moment's activity.

Like every new technological innovation, the importance of big data has burgeoned with the use of different kinds of media for personal as well as business communication. In the wake of such a growth of big data, there has also been skepticism as to what constitutes it (Sha and Carotti-Sha 2016). As mentioned above, the most common distinction by which big data can be characterized is that it cannot be stored using traditional data structures. Further the quantity of big data keeps on increasing as action takes place in real time. So, when an aircraft engine runs, then it generates real-time data over every moment that it is running. Such a dynamic dimension, as well as its profligacy, is something that distinguishes the nature of big data generation from standard data.

In tandem with this, over the last decade, there has been an exponential growth in the reliance on customer-generated data in every industry and every walk of life. In a world where instantaneous access and response to data directly influence prospects of a company, sales of business intelligence software have far outstripped sales of traditional software. Just like human capacity is being replaced by software agents, similarly there is a noticeable shift from reliance on stored data onto accessing and processing real-time data. Such real-time data is also getting generated as customers engage in consuming products and sellers are eager to curate these experiences so that they can be mimicked to increase familiarity and thus the custom of the consumer. The latter in most situations is voluminous, and as it is being generated through various sources, it requires nonstandard mechanisms of processing. So traditional formats of storing and analysis using RDBMS are increasingly found to be wanting in their capacities (Gandomi and Haider 2015). According to Davenport (2014), big data is going to play a critical role in among others “every industry that uses machinery, every industry that has physical facilities, and every industry that

involves money.” Obviously, the manufacturing industry is certainly going to be an environment which will be generating substantial amounts of data as it is no exception to the characteristics mentioned by Davenport (*ibidem*). The manufacturing industry is likely to be critically impacted upon by the presence of a great deal of data that would need to be processed using NoSQL.

Data Characteristics

The diagram in Fig. 1 is aimed to illustrate the voluminous nature of big data and the ensuing types of data structures that are compatible with it. The diagram illustrates the increasingly unstructured nature of growth of big data.

From a management perspective, the advent of big data has made it impossible to think of businesses in the same way as in the past. Whereas traditional approaches have used analytics to understand and fine-tune processes to keep management informed while alerting them to anomalies (business intelligence is driven by the idea of “exception reporting”). In contrast big data has flipped such an analytical orientation on its head. The central view of big data is that the world and the data that it describes are in a state of change and flux; those organizations that can recognize and react quickly have the upper hand in this space. The sought-after business and IT capabilities are discovery and agility, rather than stability (Davenport 2014). Table 1 below projects a few

key differences between big data and traditional analytics.

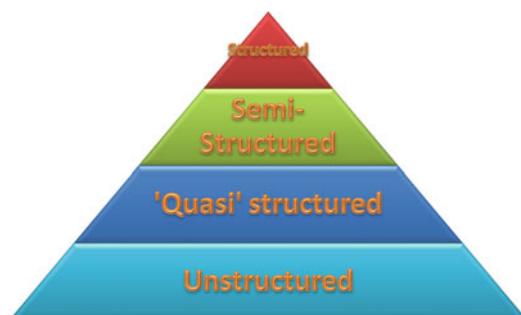
The dimension of primary purpose of big data in Table 1 reveals how the entire industrial world is reorienting itself with regard to customer data. The traditional information management approach has been to use analytics to cater to better internal decision-making through creation of reports and presentations that advise senior executives on internal decisions. In contrast data scientists are today involved in data-driven products and services through the creation of customer facing and customer touching applications.

Manufacturing Industry Transaction Processing

Given that discovery and agility are drivers that would enable manufacturing organizations to innovate and differentiate, data or big data would be at the heart of these endeavors. So in the manufacturing industry, not only manufactured products but also manufacturing equipment would increasingly contain data-producing sensors. Ability of devices that carry out, for instance, machining, welding, and robotic functions, would be reporting on their own performance and need for service. Being networked, most of these devices can be monitored and controlled from a central control hub.

Value addition in the manufacturing industry could also come about through connections with big data-fueled supply chains that would ensure supplies are available to support manufacturing while at the same time ensuring that only optimal amounts of goods are manufactured. Admittedly, discrete manufacturing companies have addressed this issue, whereas there is still a distance to go for process- or flow-based manufacturing like in the oil and natural gas sectors.

Foresight of troubleshooting products going forward along with the ability to outline expectations is critical in the manufacturing industry. Evidently, such foresight is based on accessing and evaluating a great deal of data (Marr 2016). Some of this is user generated, and some of this



Big Data Application in Manufacturing Industry,
Fig. 1 Big data growth is increasingly unstructured.
(Source: Adapted from EMC Education Services (2015))

Big Data Application in Manufacturing Industry, Table 1 Big data and traditional analytics

Dimensions	Big data	Traditional analytics
Type of data	Unstructured formats	Formatted in rows and columns
Volume of data	100 terabytes to petabytes	Tens of terabytes or less
Flow of data	Constant flow of data	Static pool of data
Analysis methods	Machine learning	Hypothesis based
Primary purpose	Data-driven products	Internal decision support and services

Source: Adapted from Davenport (2014).

Big Data Application in Manufacturing Industry, Table 2 Types of NoSQL data stores

Type	Typical usage	Examples
Key-value store – a simple data storage system that uses a key to access a value	Image stores Key-based file systems Object cache Systems designed to scale	Berkeley DB Memcached Redis Riak DynamoDB
Column family store – a sparse matrix system that uses a row and a column as keys	Web crawler results Big data problems that can relax consistency rules	Apache HBase Apache Cassandra Hypertable Apache Accumulo
Graph store – For relationship intensive problems	Social networks Fraud detection Relationship-heavy data	Neo4j AllegroGraph Bigdata (RDF data store) InfiniteGraph (objectivity)
Document store – Storing hierarchical data structures directly in the database	High-variability data Document search Integration hubs Web content management Publishing	MongoDB (10Gen) CouchDB Couchbase MarkLogic eXist-db Berkeley DB XML

Source: Adapted from McCreary and Kelly (2013).

is generated by intelligent agents such as Internet of Things (IoT) nodes.

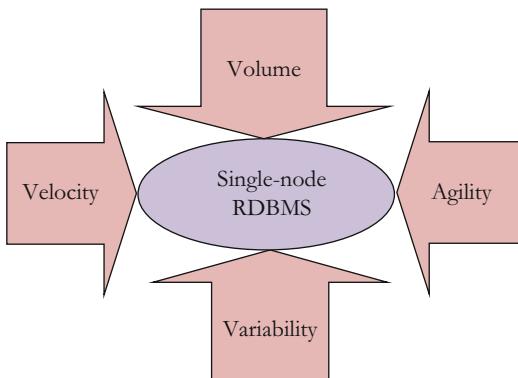
Analytics and Decision Support Requirements

As we all know, SQL is an acronym for Structured Query Language which is used to query data stored in rows and columns of databases. So SQL is used to create, manipulate, and store data in traditional formats with fixed rows and columns. However, a San Francisco-based developer group interested in issues surrounding scalable open-source databases first coined the term NoSQL (McCreary and Kelly 2013). In essence, NoSQL may be defined as:

NoSQL is a set of concepts that allows the rapid and efficient processing of data sets with a focus on performance, reliability, and agility. (McCreary and Kelly 2013; pg 4.)

It is clear from Table 2 above that there are a range of areas where NoSQL may be deemed to be a more reliable tool to process data in comparison to standard SQL. So considering data sources, key-value stores provide accommodation for a wide range of usage areas that are commonplace in the way data seems to get generated on different platforms.

Just like customer relationship management requires a completely different perspective of data storage on customers, similarly NoSQL storage envisages a change in the mindsets of data



Big Data Application in Manufacturing Industry,
Fig. 2 Impact of NoSQL business drivers. (Source: Adapted from McCreary and Kelley (2013))

handling. The diagram in Fig. 2 below projects this altered perspective.

In Fig. 2 above, the different business drivers, viz., volume, velocity, variability, and agility, are likely to apply gradual pressure on standard single CPU systems that could eventually lead to surfacing of cracks in the system's ability to handle increased processing expectations. In a real-world scenario of the manufacturing context, large amounts of data on production, of materials handling, and of supply chains all would be arriving fairly rapidly. Volume and velocity refer to the quantity and the speed at which such data would be getting generated, and so there would be increasing need to cope with these heightened requirements within restricted time horizons. Today's successful manufacturing companies would need to swiftly access data from sources that are not necessarily reducible into rows and columns. So, there may be recorded conversations, there may be Facebook posts, and there may be video recordings that would need to be considered by the manufacturing systems to make sure that business intelligence is sufficiently factored in to the manufacturing system to then cater to the continually changing expectations of the clientele. As a matter of fact, customer expectations are so quick changing that a manufacturing company can very easily find itself forced out of an industry if competitor application of NoSQL is more effective and insightful and leads to a better

match with customer expectations. The range of data types in big data is what is captured by variability of the data sets. Finally, for many manufacturing organizations, success in the market would be governed by how quickly they are able to respond to customer expectations. Such speed of response, represented by agility factors, would be reliant on the speed of successfully processing the different types of data that are going into the system.

Organization Readiness and Affordability

Manufacturing organizations may not be technologically or with regard to existing human capacity be able to modernize sufficiently quickly and integrate large volumes of data that is produced within the industrial environment. So there needs to be capacity that should be in place to accept and process these data sets. The approach that would be most appropriate would be to gradually phase in capacity to process big data. In the context of manufacturing data, organizations need to handle data using RDBMS, big data involving various formats as well as hybrid data. Using NoSQL to process big data is undoubtedly the most appropriate option for manufacturing organizations.

As an example, the levels of cost reduction achieved through big data technologies are incomparable to other alternatives in the present context. As a matter of fact, MIPS (millions of instructions per second – that is how fast a computer system crunches data) and terabyte storage for structured data are now most cheaply delivered through big data technologies like Hadoop clusters. For instance, a small manufacturing company considering storage of 1 terabyte for a year was £28 k for a traditional RDBMS, £3.9 k for a data appliance, and only £1.6 k for a Hadoop cluster – and the latter had the highest speed of processing data under most circumstances (Davenport 2014). Clearly, the cost-benefit trade-offs of not acquiring big data capabilities could potentially manifest in losing competitive advantage in the manufacturing industry.

Data Privacy and Security

As more data is available, stored in non-relational databases accessible online, and increasingly shared with third parties, the risk of data breaches also increases. Big data thus raises a number of privacy and security questions related to the access, the storage, and the usage of personal/user-related data. In the context of manufacturing companies, they share data on, among others, materials and processes with third-party organizations such as supply chain partners. So there will continue to be risks of exposure of sensitive identity attributes and other confidential information to the wider public. Additionally, for manufacturing organizations, data breaches may result in brand damages (i.e., loss of customers and partners' trust/loyalty, loss of intellectual property, loss of market share, and legal penalties and fines in case of noncompliance with privacy regulation). To deal with the abovementioned privacy risks, NoSQL has all the inbuilt capabilities of RDBMS (like GRANT, REVOKE, COMMIT, ROLLBACK for transactions). Therefore NoSQL is intrinsically suitable for encrypting data, access control, and data anonymization.

There are a couple of types of data that are sought after by external competitors of manufacturing companies. One type is raw data which is accessed by adversaries to compromise the interpretation/analysis process, e.g., injecting false data into raw data or stealing a large volume of sensitive (financial/identity) data. The second type of data that is accessed by adversaries is the different data sets that have already been analyzed as well as the actionable intelligence that analysts may have extracted from the data.

Conclusions

The discussion so far has touched upon various facets of data storage in general and big data in particular for manufacturing companies. Considering the future growth in both internal and external data that is likely to be generated, it follows that choosing NoSQL storage would be

an imperative. Circumstances of manufacturing companies would vary in the need of standard traditional RDBMS processing as well as NoSQL or hybrid data processing requirements. Hadoop, as a platform comprising of HIVE, enables NoSQL to be able to deal with all three requirements of manufacturing organizations. Considering exemplar costs, privacy, volume, variability, and agility requirements of the present and future contexts, it is clear that NoSQL is an inevitable necessity of manufacturing organizations. However, exponentially growing volume, variability, and agility of data requirement is a reality that all manufacturing companies need to cope with in order to survive in an increasingly data-driven industry where capacity to predict may make the difference between success and failure.

References

- Davenport TH (2014) Big data @work: dispelling the myths, uncovering the opportunities. HBR Press
EMC Education Services, ed. (2015) Data science & big data analytics. Wiley
Gandomi A, Haider M (2015) Beyond the hype: big data, concepts, methods, and analytics. Int J Inf Manag 35(2):137–144
Marr B (2016) Big data in practice. Wiley
McCreary D, Kelly A (2013) Making sense of NoSQL. Manning
Sha XW, Carotti-Sha G (2016) Big data. AI & Society, 1–4

Big Data Architectures

Rodrigo N. Calheiros
School of Computing, Engineering and Mathematics, Western Sydney University, Penrith, NSW, Australia

Definitions

Big data architectures comprise an abstract view of systems that enable big data. It specifies the role of diverse components of the system, their

behavior, and how they interact with each other. Examples of roles of components include batch processing (i.e., handling the volume aspect of big data), stream processing (handling velocity), data management (handling variety) and modules enabling security, processing across geographically distributed locations, and other specific functions. Software architectures are not concerned on how modules are realized (implemented); thus, it is possible that two different realizations of the same architecture might be built with very different technology stacks. Specification of a software architecture helps in the definition of a blueprint containing the long-term vision for the future system and can help in directing development efforts and technical decisions during the system implementation.

Overview

According to Len Bass, “The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both” (Bass 2013). Therefore, rather than being a detailed description of how a system is built, a software architecture describes the diverse building blocks of the system and the relationship between them, keeping aside implementation details that, even if changed, do not change the vision described in the architecture.

The software architecture of a system is an ideal manner to communicate key design principles of the system, system properties, and even the decision-making process that led the system architecture to attain a certain shape.

The more complex the system, the bigger the need for a comprehensive documentation of its architecture. Thus, in the realm of big data, where there is a need to process large quantities of fast-arriving and diverse data on a system, it is important that a blueprint exists that documents what component of the system handles each of the aspects of big data, how these parts integrate, and how they work toward achieving their goals. This is the goal of big data architectures.

Over the years, several architectures were proposed that target the challenges of big data in dif-

ferent ways and that make different assumptions about how big data applications are structured and where the system is hosted. Although the term architecture is used in the literature to also describe implementation-specific approaches, we target here the most conceptual definition of architecture, independent of how it is realized. With this definition in mind, we review the most influential big data architectures in the next pages.

Lambda Architecture

The Lambda architecture (Marz and Warren 2015) is one of the first documented architectures for big data. It contains components that handle data volume and data velocity: it combines information from batch views and real-time views to generate results of user queries.

Conceptually, the architecture is divided in three layers: the *batch layer*, the *speed layer*, and the *serving layer* (Marz and Warren 2015). The role of the batch layer is to generate batch views that are a solution of the volume problem: the assumption in the architecture design is that, because batch views are computed from large volumes of data, it is likely that this computation will run for hours (or even days). Meanwhile, there will be still data arriving in the system (velocity) and queries being made by users. Thus, the answer needs to be built with the last available batch view, which may be outdated by hours or days.

An interesting characteristic of batch views is that they are immutable once generated. The next batch view is generated from the combination of all the data available in the system at the time the previous batch view was generated in addition to all the data that arrived since such a view was built (and that is available in the speed layer). Because batch views are continuously generated, problems with the data are corrected in the next batch view generated after the problem was detected and corrected, without requiring any other action by administrators.

To compensate staleness in the batch layer, the *speed layer* processes data, via stream computing, that arrived since the latest batch view was

generated. To be feasible, it performs incremental computation. Eventually, these views are incorporated by a new batch view and cease existing in the speed layer, thus making room for continuous incorporation of data on the system.

User interaction occurs in the serving layer, which incorporates the most recent batch view and the current views generated by the speed layer to answer to user queries. For this layer to meet its purpose of providing answer to queries in reasonable time, it needs to meet certain performance expectations, and thus data denormalization is not uncommon. Moreover, it should also support random reads on the data it contains so that answer to users can be sped up. Besides performance, other desired attributes of this layer, as pointed out by Marz and Warren, are fault tolerance and scalability (Marz and Warren 2015).

The architecture targets clusters or private clouds as its underlying architecture: a certain degree of data locality is necessary to enable efficient building of the batch views and queries by the speed layer. If data were to be distributed, extra modules would be necessary to combine local views generated in geographically distinct locations.

The Lambda architecture is general enough to be applied in many big data domains. The literature contains several realizations of the architecture, with open source and custom-made tools, in different domains (Magnoni et al. 2015; Schnizler et al. 2014). Nevertheless, this architecture also has some limitations. In particular, it is designed for computations that can be performed incrementally at the speed layer and that can benefit from random access or from different computations that can anticipate different user queries. When this does not hold, the applicability of the architecture is uncertain.

Big Data Architecture Framework (BDAF)

The Big Data Architecture Framework has been proposed by Demchenko et al. (2014) and has a more data-centric perspective of big data activities than the processing-centric view of the

Lambda architecture. It also addresses a different target environment: unlike the Lambda architecture that is more suitable to a local distributed system (e.g., a cluster or a private cloud), BDAF contemplates the fact that part of the data required for computation can be geographically distributed. Thus, if user queries need to be performed over such distributed data, computation will also need to be distributed, to occur where the relevant data is.

BDAF and Lambda also diverge in the aspect of data management. BDAF envisions a data management module that is separated from processing modules. This module addresses the “variety” dimension of big data (whereas the Lambda architecture addresses velocity and volume). It encompasses capabilities of data transformation, modeling, curation, and access. It also acknowledges the need to keep different views of data (i.e., different transformations over the same data) to serve as input to different analytics tasks.

Finally, BDAF also contains modules addressing security and coordination of distributed components, which is not necessary in the Lambda architecture because of the different target environment (centralized vs. distributed). This module is in charge of aspects such as federation (i.e., seamless access to resources that may be physically located in different infrastructure, thus avoiding the need for users to login or access each location individually), security, and interoperation (so the same operations can be carried out in different locations even if details on how to access such operations differ across locations).

In terms of applications, BDAF envisions different uses for the architecture. Some applications might just require batch processing, others just streaming, and others a combination of both. More complex programming such as workflows are also envisioned and supported in the architecture.

In a later publication (Demchenko et al. 2016), authors of BDAF mapped components of the architecture to the context of different big data scientific projects, demonstrating that the components of the proposed architecture match requirements of real systems.

In summary, BDAF is an architecture more concerned with data management aspects of big data and thus complements well the Lambda architecture, which has a bigger emphasis on processing capabilities with strong assertions on how data should be presented. However, its design is more complex to be able to address inherent issues with its complex target infrastructure.

BASIS

BASIS (Costa and Santos 2016) is a big data architecture that has been designed with a specific domain in mind, namely, smart cities. Nevertheless, the principles and design choices make it suitable to general domains as well.

Given its target context of smart cities, it is assumed that there are several sources of data to the system that needs to be transformed before being analyzed. This data enters the system via processing modules that include not only batch and stream (handling volume and velocity, as in the previous architectures) but also specific modules for data integration (handling variety). Other necessary components are APIs supporting ingestion of data from sensors and other particular devices and connectors to databases (enabling data to be persistently stored).

In its storage layer, BASIS encompasses modules supporting different types of data storage: file storage (called landing zone), distributed storage (for large volume), and three special-purpose storage modules, handling operational data, administrative data, and data to be made publicly available.

On top of this processing and storage infrastructure, the architecture contains a layer supporting arbitrary big data analytics tasks, which are accessed by applications that lay in the layer above, the topmost.

An implementation of the architecture is also presented by the authors to demonstrate the viability of the architecture. In terms of capabilities, BASIS is very similar to BDAF, but also differentiates itself in some aspects. The first one is that requirements for cross-site coordination and security are of smaller concern in BASIS,

as it is not design for the same level of distribution targeted by BDAF. However, BASIS has a stronger emphasis in enabling analytics as a service that can be composed by applications to provide specialized analytics to users, which is the main differential of BASIS compared to the other discussed architectures.

SOLID

SOLID (Cuesta et al. 2013) is a big data architecture for clusters that has been developed with the goal of prioritizing the variety dimension of big data over volume and velocity. As a result, many architectural decisions are made that differ significantly from the other approaches described so far.

The first key difference regards the expected data model. SOLID models data as *graphs*. The second key difference is that it mandates a specific data model, namely, RDF (Resource Description Framework (Manola et al. 2004)) to be used in the architecture.

Conceptually, SOLID is similar to the Lambda architecture with a few more components. The *merge layer* is responsible for batch processing (thus handling the volume aspect of big data). Similar to the case of Lambda architecture, it generates immutable views from all the data available until the moment the view is generated and stores the result in a database.

The *online layer* is conceptually equivalent to Lambda's speed layer, handling big data's velocity. It stores data received since the last dump to the merge layer. The *service layer* receives user queries and issues new queries to the online and index layers to build the answer to the query, and thus this is equivalent to Lambda's serving layer. Notice that, different from the Lambda architecture's serving layer, the online layer does not precompute views to be used to answer queries; it builds the answers on demand based on received requests.

Besides the three layers that are equivalent to Lambda layers, two other layers are present in SOLID: the first extra layer is the *index layer*, which generates indices that speed up

the access to the data items generated by the merge layer. The second one is the *data layer*, which stores data and metadata associated with the architecture, which is generated by the merge layer.

Quality-Centric Big Data Architecture for Federated Sensor Services

Ramaswamy et al. (2013) proposed a cloud-based big data architecture to support the Internet of Things (IoT) and all the data generated by IoT sensors. The key concern in the design of the architecture was enabling quality data to be obtained, stored, and accessed by end users. The objective of this vision is reducing the investment required to run IoT sensor networks: if the data of such sensors are valuable (timely and accurate) and it can be accessed by end users as a service, the cost of installing and maintaining such networks can be amortized among many users over time.

Data quality aspects considered by the architecture are not only performance metrics (latency, throughput) but also accuracy and variety. Moreover, the architecture requires components that can discover and add data feeds from sensors to the architecture. If this is the case, transformation in the data of added sensors may be necessary to conform with expectations from the architecture. Finally, mechanisms for billing are necessary, as one of the goals of the architecture is making the platform economically viable.

Conceptually, the platform contains not only modules for batch processing and stream analysis (such as Lambda's batch and speed layers, respectively) but also other components, pricing, network services, storage services, and sensor services, which virtualize and provide quality attributes to the corresponding elements, and also other modules with more specialized functions.

One of such specialized modules is the *DQ Monitor*. This module is responsible for managing the quality of data emerging from sensors (and therefore it communicates with the sensor service module). The *DQ Service Catalog* re-

ceives access from *brokers*, which act on behalf of users, to provide information about available sensors and corresponding data feeds.

B

M3Data

M3Data (Ionescu et al. 2016) is a cloud-based big data architecture that differs from the above architectures by envisioning a cycle for data processing that applies ETL (extract, transform, load) operations, similarly to classic business intelligence applications.

Some architectural components from M3Data are designed to support and interact with cloud infrastructures, whereas other components are designed to support actual operation on the data. The first group includes the *control layer*, which is responsible for operations such as installation and deployment of relevant software packages, and other self-explained layers such as *security layer*, *communication layer*, *processing layer*, *notification layer*, and *data transfer layer*.

The *flow control* module enables description and execution of complex dataflows, where users can describe operations to be carried out on data (such as cleansing, processing, transferring, and so on). Most importantly, this can be done graphically via a *dataflow studio* software.

M3Data also supports diverse data sources, including proprietary databases, file systems, FTP and email servers, and third-party applications. New data sources can be discovered and accessed, and its data can be indexed for future use. In fact, indexing and searching are important aspects of the architecture, with dedicated components for these tasks.

Another highlight of M3Data is support for autonomic computing, where the IT infrastructure is self-managed and can autonomously react to failures and other anomalies.

Summary of Architectures

Table 1 summarizes the key points of the evaluated architectures. It can be noticed that they

Big Data Architectures, Table 1 Summary of big data architectures

	Key application domain	Target infrastructure	Module for volume?	Module for velocity?	Module for variety?	Infrastructure-level modules?	Application-level modules?	Design highlights
Lambda (Marz and Warren 2015)	Queries over stream and stored data	Clusters	Yes	Yes	No	No	No	Immutable views Implemented by third parties
BDAF (Demchenko et al. 2014)	Geo-distributed queries	Geo-distributed clouds	Yes	Yes	Yes	Yes	Yes	Data-centric Support for multiple programming models
BASIS (Costa and Santos 2016)	Smart cities	Clusters	Yes	Yes	Yes	No	Yes	Supports different types of data stores Supports analytics as a service
SOLID (Cuesta et al. 2013)	Real-time systems	Clusters	Yes	Yes	Yes	No	No	Strong support for “variety” Module for indexing
Quality-centric (Ramaswamy et al. 2013)	Internet of Things (IoT)	Clouds	Yes	Yes	Yes	Yes	No	Modules to discover and aggregate data from new sensors
M3Data (Ionescu et al. 2016)	Data manipulation and querying	Clouds	Yes	Yes	Yes	Yes	Yes	Support for ETL operations Autonomous

differ in target domains, platforms, support for different features, and main features. It is evident that there is no one-size-fits-all solution for big data architectures, and the answer to the question of which one is the best depends on individual requirements.

Examples of Applications

A key application of big data architectures regards initial blueprints of big data solutions in a company or specific domain. In early stages of adoption, architects need to understand the re-

quirements of the system being designed, where it will be applied, what are the data sources, and what transformations and queries will occur in the data.

Once the above is identified, an architect needs to *select the big data architecture whose design and goals are closer to the system being conceived*. During this stage, two situations can be observed. In the first situation, it can be determined that some functionalities are missing in the candidate architecture. If this is the case, new capabilities will need to be designed and incorporated to the final system.

In the second case, the architect might identify that some elements of the architecture antagonize with needs/requirements of the system under development. In this case, a careful assessment needs to be carried out to identify the risks and costs involved in circumventing the antagonistic characteristics. Solutions to such situation can require complex logic and brittle software systems that may fail in achieving their goals. Therefore, architects should avoid selecting candidate architectures with antagonistic capabilities in relation to the needs of the system.

minimum knowledge about the underlying platform.

B

Cross-References

- ▶ [Big Data Indexing](#)
- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Definition of Data Streams](#)
- ▶ [ETL](#)
- ▶ [Graph Processing Frameworks](#)
- ▶ [Hadoop](#)
- ▶ [Large-Scale Graph Processing System](#)
- ▶ [Rendezvous Architectures](#)
- ▶ [Streaming Microservices](#)

Future Directions for Research

- *Leveraging emerging computing platforms in new architectures.* Many of the discussed architectures were designed to benefit from public cloud computing platforms. This requires addition of new components on the architecture, but this decision empowers applications with elasticity and larger scale than achievable by cluster and private clouds. We are witnessing the emergence of new platforms such as edge computing and fog computing, which offer a compromise between local-only and cloud-only solutions. To enable big data architectures to benefit from such platforms, research is needed to identify the extra challenges incurred by these platforms and how it can be addressed in emerging architectures.
- *User-centric capabilities.* Existing architectures aim to solve abstract or specific problems with little or no focus on end users. Some architectures are data-centric, in which most of its components exist to operate on the data. Other architectures contain analytics and application modules, but none of them focus in the user experience. Therefore, research is needed to facilitate the interaction between end user and the system, so non-ICT specialist users can easily describe their desired operations, visualization, and reporting with

References

- Bass L (2013) Software architecture in practice, 3rd edn. Addison-Wesley, Reading
- Costa C, Santos MY (2016) BASIS: a big data architecture for smart cities. In: Proceedings of the 2016 SAI computing conference, IEEE, London, 13–15 July 2016
- Cuesta CE, Martínez-Prieto MA, Fernández JD (2013) Towards an architecture for managing big semantic data in real-time. In: Drira K (ed) Software architecture. ECSA 2013, Lecture notes in computer science, vol 7957. Springer, Berlin/Heidelberg
- Demchenko Y, de Laat C, Membrey P (2014) Defining architecture components of the Big Data Ecosystem. In: Proceedings of the 2014 international conference on collaboration technologies and systems (CTS), IEEE, Minneapolis, 19–23 May 2014
- Demchenko Y, Turkmen F, de Laat C et al (2016) Cloud based big data infrastructure: architectural components and automated provisioning. In: Proceedings of the 2016 international conference on high performance computing & simulation (HPCS 2016), IEEE, Innsbruck, 18–22 July 2016
- Ionescu B, Ionescu D, Gadea C et al (2016) An architecture and methods for big data analysis. In: Balas VE et al (eds) Soft computing applications, advances in intelligent systems and computing, vol 356. Springer, Berlin/Heidelberg
- Magnoni L, Suthakar U, Cordeiro C et al (2015) Monitoring WLCG with lambda-architecture: a new scalable data store and analytics platform for monitoring at petabyte scale. J Phys Conf Ser 664:052023
- Manola F, Miller E, McBride B (eds) (2004) RDF primer. W3C recommendation. Available at <https://www.w3.org/TR/rdf-primer/>. Accessed February 2018.

- Marz N, Warren J (2015) Big Data: principles and best practices of scalable realtime data systems. Manning, Stamford
- Ramaswamy L, Lawson V, Gogineni SV (2013) Towards a quality-centric big data architecture for federated sensor services. In: Proceedings of the 2013 IEEE international congress on Big Data (BigData Congress), IEEE, Santa Clara, 27 June–2 July 2013
- Schnizler F, Liebig T, Marmor S et al (2014) Heterogeneous stream processing for disaster detection and alarming. In: Proceedings of the 2014 IEEE international conference on Big Data (Big Data), IEEE Computer Society, Washington, DC, 27–30 Oct 2014

Big Data Availability

- ▶ [Data Replication and Encoding](#)

Big Data Benchmark

- ▶ [TPC-DS](#)

Big Data Benchmarks

- ▶ [Analytics Benchmarks](#)

Big Data Deep Learning Tools

Nur Farhana Hordri^{1,2}, Siti Sophiayati Yuhaniz¹, Siti Mariyam Shamsuddin², and Nurulhuda Firdaus Mohd Azmi¹

¹Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia

²UTM Big Data Centre, Universiti Teknologi Malaysia, Johor, Malaysia

Definition

Deep Learning or also known as deep structured learning or hierarchical learning is a part of a broader family of Machine Learning methods

based on learning data representations (Bengio et al. 2013). Giving that Hordri et al. (2017) have systematically reviewed that the features of the Deep Learning are a hierarchical layer, high-level abstraction, process a high volume of data, universal model, and does not overfit training data. Deep Learning can be applied to learn from labeled data if it is available in sufficiently large amounts; it is particularly interesting to learn from large amounts of unlabeled/unsupervised data (Bengio et al. 2013; Bengio 2013), so it is also interesting to extract meaningful representations and patterns from Big Data. Since data keeps getting bigger, several Deep Learning tools have appeared to enable efficient development and implementation of Deep Learning method in Big Data (Bahrampour et al. 2015). Due to Deep Learning having different approaches such as convolutional neural network (CNN) and recurrent neural network (RNN), these Deep Learning tools are going to optimize different aspects of the development of Deep Learning approaches. The list of the Deep Learning tools includes but not limited to Caffe, Torch7, Theano, DeepLearning4j, PyLearn, deepmat, and many more. However, in this paper we will only focus on three tools such as Caffe, Torch7, and Theano since most of the researchers are using these tools according to the 2015 KDnuggets Software Poll.

Overview

Big Data generally refers to data that exceeds the typical storage, processing, and computing capacity of conventional databases and data analysis techniques. While Big Data is growing rapidly in the digital world in different shapes and sizes, there are constraints when managing and analyzing the data using conventional tools and technologies. Most conventional learning methods use shallow-structured learning architectures. Since the constraints are not an ordinary task, they require a collaboration of both developments of advanced technologies and interdisciplinary teams. Hence, Machine Learning techniques, together with advances in available computational power, have come to play an important role in Big

Data analytics and knowledge discovery (Chen and Lin 2014). Today, Deep Learning becoming a trend due to its techniques that use supervised and/or unsupervised strategies to automatically learn hierarchical representations in deep architectures for classification (Bengio and Bengio 2000; Boureau and Cun 2008).

One of the main tasks associated with Big Data Analytics is retrieving information (National Research Council 2013). Efficient storage and retrieval information are a growing problem in Big Data, primarily because of the enormous quantities of data such as text, image, video, and audio being collected made available across multiple domains. Instead of using raw inputs for indexing data, Deep Learning can be used to generate high-level abstract data representations to be used for semantic indexing. These representations can expose complex associations and factors (especially when the raw input is Big Data), which leads to semantic knowledge and understanding. While Deep Learning in providing an understanding of semantic and relational of the data, a vector representation (corresponding to the extracted representations) from data instances would provide faster searching and obtain information. With Deep Learning one can leverage unlabeled documents (unsupervised data) to have access to a much larger amount of input data, using a smaller amount of supervised data to improve the data representations and make it more relevant in learning task and specific conclusion.

Chen and Lin (2014) has discussed on how Deep Learning has also been successfully applied in industry products that take advantage of the large volume of digital data. Google, Apple, and Facebook are one of the companies that collect and analyze massive amounts of data everyday which aggressively make Deep Learning push forward to each related project. As in Efrati (2013), by utilizing Deep Learning and data collected by Apple services, the virtual personal assistant in iPhone, Siri, has offered a wide variety of services including weather reports, sports news, answer to user's questions, and reminder, while Google uses Deep Learning algorithms for massive chunks of messy data which are

obtained from the Internet for Google's translator, Android's voice recognition, Google's street view, and image search engine (Jones 2014). Microsoft also uses language translation in Bing voice search. IBM joins forces to build a brain-like computer by using Deep Learning to leverage Big Data for competitive advantage (Kirk 2013).

In summary, we could not find any related works on reviewing Deep Learning tools in Big Data analytics. Therefore, the purpose of the paper is to review the current top-listed tools of Deep Learning in Big Data applications. Findings may assist other researchers in determining the potential tools that going to be used in their study. The remainder of this paper is divided into following parts: in section “[Tools of Deep Learning Big Data](#),” we summarized each of the top three tools for Deep Learning. Moreover, we discussed the recent applications, languages, environments, and libraries for each tool. This is to ensure a good understanding on the concept of the common Deep Learning tools for Big Data that have been done over the years. Consequently, in section “[Comparative Study of Deep Learning Tools](#),” we created a comparative table of the properties of tools that have been discussed in section “[Tools of Deep Learning Big Data](#).” Finally, we have concluded this paper in the last section which is in section “[Future Directions for Research](#).”

Tools of Deep Learning Big Data

This section describes the foundation of this review by discussing several of tools of Deep Learning that have been applied with Big Data.

Caffe. Convolutional Architecture for Fast Feature Embedding or also known as Caffe is a Deep Learning framework made with expression, speed, and modularity in mind (Tokui et al. 2015). Caffe was originally developed at UC Berkeley by Yangqing Jia which is open sourced under BSD license. The code is written in clean, efficient C++ with CUDA used for GPU computation and nearly complete well-supported bindings to Python/Numpy and MATLAB (Jia et al. 2014; Tokui et al. 2015; Kokkinos

2015). Caffe becomes a preferable framework for research use due to the careful modularity of the code and the clean separation of network definition from actual implementation (Tokui et al. 2015). Several different types of Deep Learning architecture that Caffe support as mentioned in Caffe Tutorial (2018) are image classification and image segmentation. Caffe supports CNN, Regions with CNN (RCNN), long short-term memory (LSTM), and fully connected neural network designs.

According to Jia et al. (2014), Caffe stored data and communicates data in four-dimensional arrays called blobs. Blobs provide a unified memory interface, holding batches of images, parameters, or parameter updates. The data is transferred from one blob to another blob in the next layer. Based on Chetlur et al. (2014), in Caffe each type of model operation is encapsulated in a layer. Layer development comprises declaring and implementing a layer class, defining the layer in the protocol buffer model scheme, extending the layer factory, and including tests. These layers take one or more blobs as input and yield one or more blobs as output. Each layer has two key responsibilities for the operation of the network. There are (i) a *forward pass* that takes the inputs and produces the output and (ii) a *backward pass* that takes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers. After that, Caffe does all the bookkeeping for any directed acyclic graph of layers, ensuring correctness of the forward and backward passes. A typical network begins with a data layer which is run on CPU or GPU by setting a single switch. The CPU/GPU switch is seamless and independent of the model definition.

Caffe is able to clear access to deep architectures which can train the models for fastest research progress and emerged the commercial applications. In 2015, Ahmed et al. (2015) has implemented deep network architecture using the Caffe Deep Learning framework which the research adapted various of layers from the framework and wrote their own layers that are specific to the deep network architecture. The convergences time for training the network is approxi-

mately 12–14 h. The architecture is implemented in NVIDIA GTX780 and NVIDIA K40 GPUs. Also, in 2015, Kokkinos (2015) has evaluated the performance of Caffe, cuDNN, and CUDA-convnet2 using the layer configurations which are commonly used for benchmarking convolutional performance and quoting average throughput for all these layers. This performance portability is crossed GPU architecture with Tesla K40. Tokui and co-workers (2015) have introduced Chainer, a Python-based, stand-alone, open-source framework for Deep Learning models (Caffe Tutorial 2018). Tokui claims that Chainer provides a flexible, intuitive, and high-performance means of implementing a full range of Deep Learning models. He then compared Chainer with Caffe using various CNNS. Based on the experiment, since Caffe is written in C++ and Chainer is written in Python, the total time required for trial-and-error testing in implementing and tuning new algorithms is likely to be similar for both frameworks. Still in 2015, Kokkinos combined a careful design of the loss for boundary detection training, a multi-resolution architecture and training with external data to improve an accuracy of the current state of the art (Chetlur et al. 2014). This detector is fully integrated into the Caffe framework and processes a 320×420 image in less than a second on an NVIDIA Titan GPU.

Torch7. Collobert et al. (2011) defined Torch7 as a versatile numeric computing framework and Machine Learning library that extends Lua (Collobert et al. 2011). By using Lua you can have a lightweight scripting language. Collobert and his team chose Lua because Lua is the only fastest interpreted language that they can find. Lua is easy to be embedded in a C application. They also chose Lua instead of Python because Lua is well-known in the gaming programmer community. Most packages in Torch7 or third-party packages that depend on Torch7 rely on its Tensor class to represent signals, images, and videos. There are eight packages in Torch7 which are (i) torch, (ii) lab and plot, (iii) qt, (iv) nn, (v) image, (vi) optim, (vii) unsup, and (viii) third-party. Lua also provides a wide range of algorithms for Deep Machine Learning and uses the scripting LuaJIT.

Torch7 is designed and developed to use Open Multi-Processing (OpenMP) directives for various operations in its Tensor library and neural network package (Collobert et al. 2011). This is due to the OpenMP which provides a shared memory CPU parallelization framework in C/C++ and Fortran languages on the almost operating system and compiler tool set. Hence, OpenMP normally requires minimal modification for integrating into an existing project. If the compiler is supported by OpenMP, Torch7 will automatically detect it and compile a high-level package that adds multi-threaded Tensor operations, convolutions, and several neural network classes. The number of threads to be used can be specified by either setting or environment variable to the desired number.

In 2015, Lane et al. (2015) have done the experiment in designing the raw feasibility of executing Deep Learning models. The paper also considered end-to-end model performance metrics of execution time and energy consumption, especially in terms of how these map to application needs. For the experiments, Torch7 has been used for training all the models in NVIDIA Tegra K1. The result of the experiments shows that Deep Learning is enabling to be ready to integrate with IOT, smartphones, and wearable systems. Shokri and Shmatikov (2015) have designed, implemented, and evaluated a practical system that enables multiple parties to jointly learn an accurate neural network model for a given objective without sharing their input datasets (Shokri and Shmatikov 2015). They used Torch7 packages as they mentioned that Torch7 has been used and extended by major Internet companies such as Facebook. Later in 2016, small towers of wooden blocks are created using a 3D game engine which the stability is randomized and renders them collapsing (Lerer et al. 2016), using Torch7 Deep Learning environment to integrate with Machine Learning framework directly into the UE4 game loop. Torch7 allowed fine-grained scripting and online control of UE4 simulations. Since Lua is the dominant scripting languages for games and many games including UE4 support, Torch7 is well-suited for game engine integration. The

simulations from Torch7 can be run faster than real time and help in large-scale training. Other than games, the author also said that Torch7 is useful for experiments in vision, physics, and agent learning.

Theano. Theano is a Python library that allows to define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently (Al-Rfou et al. 2016; Goodfellow et al. 2013). Theano aims to improve both execution time and development time of Machine Learning applications, especially Deep Learning algorithm (Bergstra et al. 2011). By describing the mathematical algorithms as in functional language and profit from fast coding, Theano makes it easier to leverage the rapid development pattern encouraged by the Python language and depending on Numpy and Scipy. Theano also provides a language that is an actual implementation for describing expressions (Sherkhane and Vora 2017). Likewise, in Bastien's paper (2012), Theano uses CUDA to define a class of n -dimensional arrays located in GPU memory with Python binding (Bastien et al. 2012). Theano can automatically compute symbolic differentiation of complex expressions, disregard the variables that are not required to compute the final output, recycle partial results to avoid redundant computations, apply mathematical simplifications, calculate operations in place when possible to minimize the memory usage, and apply numerical stability optimization to overcome or minimize the error due to hardware approximations (Al-Rfou et al. 2016).

Theano defines a language to represent mathematical expressions and manipulate them. For example, Theano represents symbolic mathematical expressions as directed, acyclic graphs (Al-Rfou et al. 2016). These graphs are also bipartite, containing two kinds of nodes: variable nodes and apply nodes. Practically, for graph inputs and outputs, usually variables and intermediate values are going to be used. During the execution phase, values will be provided for input variables and computed for intermediate and output ones. The apply node has inputs and outputs, while the variable node can be the input to several

Big Data Deep Learning Tools, Table 1 Properties of Deep Learning tools (Boureau and Cun 2008)

Property	Caffe	Torch7	Theano
Core language	C	Lua	Python
CPU	✓	✓	✓
Multi-threaded CPU	✓ BLAS	✓ Widely used	✓ BLAS, conv2D limited OpenMP
GPU	✓	✓	✓
Multi-GPU	✓ Only data parallel	✓	✗ Experimental version available
NVIDIA cuDNN	✓	✓	✓
Quick deployment on standard models	✓ Easiest	✓	✗ Via secondary libraries
Automatic gradient computation	✓	✓	✓ Most flexible (also over loops)

apply nodes but can be the output of at most one apply node. Since variables are strongly typed in Theano, the type of these variables must be specified at creation time. By calling Python functions on variables, the user can then interact with them in a direct and natural way. Due to the computation graph is acyclic and its structure is fixed and independent from the actual data, it can be a challenge to express loops symbolically. Therefore, Theano implements a special Op called Scan to abstracts the entire loop in a single apply node in the graph. The Scan node handles the communication between the external or outer computation graph it belongs to and the internal or inner graph. The Scan is also responsible to manage the bookkeeping between the different iterations.

Glorot et al. (2011) proposed a Deep Learning approach which learns to extract a meaningful representation of the problem of domain adaptation of sentiment classifiers (Glorot et al. 2011). All algorithms were implemented using the Theano library, while in 2016, Litjens and his team (2016) introduced Deep Learning as a technique to improve the objectivity and efficiency of histopathologic slide analysis. In this work, the convolutional neural network has been trained using the open source Deep Learning libraries which are Theano. However, in this work, due to the memory limitations of the GPU, some of the sized patches performed substantially worse on patch-based accuracy.

Comparative Study of Deep Learning Tools

This section provides a comparison of Deep Learning tools as in Table 1 below. Table 1 describes the general properties of Caffe, Torch7, and Theano. According to Bahrampour et al. (2015), these tools are the top three well-developed and widely used frameworks by the Deep Learning community. However, the comparison only focused on the speed of performance of the convolutional frameworks.

Future Directions for Research

Overall, Deep Learning tools that have been discussed in the previous sections may give the guideline to the other researcher on which tools that want to use in their study. We found that each of the tools is effective when the researchers use to match their study. Nevertheless, since the data keeps getting bigger, the technology also becoming more advanced. There might be new Deep Learning tools that can outperform the current tools.

Conclusion

In contrast between conventional Machine Learning tools and Deep Learning tools, Deep Learning has the advantages of potentially providing a

solution to Big Data analysis due to its extracting complex data representations from large volumes of unsupervised data. Deep Learning tool becomes valuable tools when it comes to Big Data analytics where the analysis data involving large volumes of raw data are unsupervised and uncategorized. The tools can be applied to managing and analyzing the data in Big Data analytics problem due to its features that can learn from hierarchical representation in deep architectures for classification. Hence, in this study, we have aimed and identified the top three tools of Deep Learning that can be useful to Big Data. The tools are Caffe, Torch7, and Theano. This study also has given a comparative study on the performance of the convolutional frameworks for each tool.

Cross-References

- [Big Data Analysis Techniques](#)
- [Deep Learning on Big Data](#)
- [Languages for Big Data Analysis](#)
- [Overview of Online Machine Learning in Big Data Streams](#)
- [Tools and Libraries for Big Data Analysis](#)

Acknowledgments This work is supported by The Ministry of Higher Education, Malaysia under Fundamental Research Grant Scheme: 4F877.

References

- Ahmed E, Jones M, Marks TK (2015) An improved deep learning architecture for person re-identification. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, pp 3908–3916
- Al-Rfou R, Alain G, Almahairi A, Angermueller C, Bahdanau D, Ballas N, Bastien F, Bayer J, Belikov A, Belopolsky A, Bengio Y (2016) Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688, pp 472–473
- Bahriampour S, Ramakrishnan N, Schott L, Shah M (2015) Comparative study of caffe, neon, theano, and torch for deep learning. arXiv preprint arXiv:1511.06435
- Bastien F, Lamblin P, Pascanu R, Bergstra J, Goodfellow I, Bergeron A, Warde-Farley D, Bengio Y (2012) Theano: new features and speed improvements. arXiv preprint arXiv:1211.5590
- Bengio Y (2013) Deep learning of representations: looking forward. In: International conference on statistical language and speech processing, Springer, Berlin/Heidelberg, pp 1–37
- Bengio Y, Bengio S (2000) Modeling high-dimensional discrete data with multi-layer neural networks. In: Solla SA, Leen TK, Muller KR (eds) Advances in neural information processing systems. MIT Press, Boston, pp 400–406
- Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828. <https://doi.org/10.1109/tpami.2013.50>. arXiv: 1206.5538
- Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A, Bengio Y (2011) Theano: Deep learning on gpus with python. In NIPS 2011, BigLearning Workshop, Granada, Spain, vol. 3
- Boureau YL, Cun YL (2008) Sparse feature learning for deep belief networks. In: Advances in neural information processing systems, NIPS Press, Vancouver, pp 1185–1192
- Caffe Tutorial (2018) https://web.archive.org/web/20170405073658/https://vision.princeton.edu/courses/COS598/2015sp/slides/Caffe/caffe_tutorial.pdf
- Chen XW, Lin X (2014) Big data deep learning: challenges and perspectives. IEEE Access 2:514–525
- Chetlur S, Woolley C, Vandermersch P, Cohen J, Tran J, Catanzaro B, Shelhamer E (2014) cuDNN: efficient primitives for deep learning. arXiv preprint arXiv:1410.0759
- Collobert R, Kavukcuoglu K, Farabet C (2011) Torch7: a matlab-like environment for machine learning. In: BigLearn, NIPS workshop, no. EPFL-CONF-192376
- Efrati A (2013) How ‘deep learning’ works at Apple, beyond. Information (online). <https://www.theinformation.com/How-Deep-Learning-Works-at-Apple-Beyond>
- Glorot X, Bordes A, Bengio Y (2011) Domain adaptation for large-scale sentiment classification: a deep learning approach. In: Proceedings of the 28th international conference on machine learning (ICML-11), Washington, pp 513–520
- Goodfellow IJ, Warde-Farley D, Lamblin P, Dumoulin V, Mirza M, Pascanu R, Bergstra J, Bastien F, Bengio Y (2013) PyLearn2: a machine learning research library. stat, 1050, p. 20
- Hordri NF, Samar A, Yuhaniz SS, Shamsuddin SM (2017) A systematic literature review on features of deep learning in big data analytics. Int J Adv Soft Comput Appl 9(1):32–49
- Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia, ACM, New York, pp 675–678
- Jones N (2014) Computer science: the learning machines. Nature 505(7482):146–148
- Kirk J (2013) Universities, IBM join forces to build a brain-like computer. PCWorld. http://www.pcworld.com/article/243133/universities_ibm_join_forces_to_build_a_brain-like_computer.html

- pcworld.com/article/2051501/universities-join-ibm-in-cognitive-computing-researchproject.html
- Kokkinos I (2015) Pushing the boundaries of boundary detection using deep learning. arXiv preprint arXiv:1511.07386
- Lane ND, Bhattacharya S, Georgiev P, Forlivesi C, Kawsar F (2015) An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In: Proceedings of the 2015 international workshop on internet of things towards applications, ACM, New York, pp 7–12
- Lerer A, Gross S, Fergus R (2016) Learning physical intuition of block towers by example. In: International conference on machine learning, pp 430–438
- Litjens G, Sánchez CI, Timofeeva N, Hermse M, Nagtegaal I, Kovacs I, ..., Van Der Laak J (2016) Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Sci Rep* 6:26286
- National Research Council (2013) Frontiers in massive data analysis. The National Academies Press, Washington, DC. http://www.nap.edu/openbook.php?record_id=18374
- Sherkhane P, Vora D (2017) Survey of deep learning software tools. In: 2017 International conference on data management, analytics and innovation (ICDMAI), IEEE, Pune, pp 236–238
- Shokri R, Shmatikov V (2015) Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, ACM, New York, pp 1310–1321
- Tokui S, Oono K, Hido S, Clayton J (2015) Chainer: a next-generation open source framework for deep learning. In: Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS), vol 5

LMI is, it can be referred to the design and use of AI algorithms and frameworks to analyze data related to labor market (*aka* labor market information) for supporting policy and decision-making (see, e.g., UK Commission for Employment and Skills 2015; UK Department for Education and Skills 2004).

Classification system or taxonomy in the field of labor market refers to a taxonomy or a graph that organizes jobs into a clearly defined set of groups according to the tasks and duties undertaken in the job, as in the case of the International Standard Classification System ISCO (Organization IL 2017) and the US classification system O*NET (U.S. Department of Labor/Employment & Training Administration 2017). Recently, such systems have been improved to take into account also skills, competences, and qualifications, as the case of the European classification system ESCO (European Commission 2017).

KDD stands for knowledge discovery in databases. It is a framework firstly introduced by Fayyad et al. (1996) for extracting knowledge from huge mass of data. It specifies a number of steps – from data collection to data visualization – that form a guidance for any applications that intend to retrieve knowledge from data, as in the case of Big Data applications as well.

Web job vacancy is job advertisement composed of two text fields: a title and a full description. The former shortly summarizes the job position, while the full description field usually includes the position details and the relevant skills that an employee should hold.

Big Data Enables Labor Market Intelligence

Mario Mezzanica and Fabio Mercurio
Department of Statistics and Quantitative Methods – CRISP Research Centre, University of Milan Bicocca, Milan, Italy

Definitions

Labor market intelligence (LMI) is a term that is emerging in the whole labor market community, especially in the European Union. Although there is no unified definition of what

Overview

In the last years, several forces and factors have dramatically changed the nature and characteristics of the labor market, in both advanced and developing countries. Technical progress, globalization, and the reorganization of the production processes have radically modified the demand for certain skills, intensifying the need for continued

training, especially in those jobs that are highly specialized or heavily affected by digitalization.

On one hand several jobs are disappearing, while new jobs are emerging, some of which are simply a variant of existing jobs, others are genuinely new jobs that were nonexistent until few years ago. On the other hand, the quantity and quality of the demand for skills and qualifications associated to the new labor market have changed dramatically. New skills are needed not only to perform new jobs, but also the skill requirements of existing jobs have changed considerably.

The Need for LMI. In such a dynamic scenario, the problem of monitoring, analyzing, and understanding these labor market changes (i) timely and (ii) at a very fine-grained geographical level has become practically significant in our daily lives. *Which occupations will grow in the future and where? What skills will be demanded the most in the next years?* Those are just few questions with which economists and policy makers have to face with (see, e.g., the recent work by Frey and Osborne (2017) about occupations that might disappear due to the digitalization). Here, a key role is played by Big Data concerning labor market (e.g., job advertisements, cv posted on the Web, etc.). This data needs to be collected, organized, and manipulated for enabling a real-time monitoring and analysis of labor market dynamics and trends. In this way, labor market

needs can be analyzed without the time lags found in traditional administrative and survey data sources that may require several months for being available. In addition, a real-time analysis of Web labor market represents an anytime snapshot of the market demands, providing a useful source of information about specific job requirements, the offered contracts, requested skills (both hard and soft), the industry sector, etc., which are not covered in any surveys.

A Motivating Example. Table 1 shows two job vacancies extracted from specialized Web sites. Even though both advertisements seek for *computer scientists*, the differences in terms of job requirements, skills, and corresponding educational levels are quite evident. The first job vacancy (A) is looking for a software developer, while the second one (B) is searching for a less specialized candidate, i.e., an ICT technician. Indeed, in the latter case, the only requested abilities required are to be able to *use* some solutions and the knowledge of a programming language (nice to have) that is usually taught in some professional high schools.

The ability to catch such differences and to classify these two distinct occupational profiles (i) timely and (ii) according to a standard taxonomy is mandatory for analyzing, sharing, and comparing the Web labor market dynamics over different regions and countries. Here, job offer

Big Data Enables Labor Market Intelligence, Table 1
CEDEFOP project (Taken from Boselli et al. 2017b)

(A) **ICT Developer.** “Looking to recruit a software developer to join our dynamic R&D team to support work on Windows-based software for current and upcoming products. A flexible, self-starting attitude is essential along with a strong motivation to learn new skills as required. The ideal candidate will have at least 2–3 years experience working in a fast and dynamic small team. Experience in Python is the key requirement for my client. A degree in computer science/computer engineering is preferable, but other engineering/science graduates will be considered if they have software development experience.” [ISCO 2512: Software Developer]

Workplace: Milan

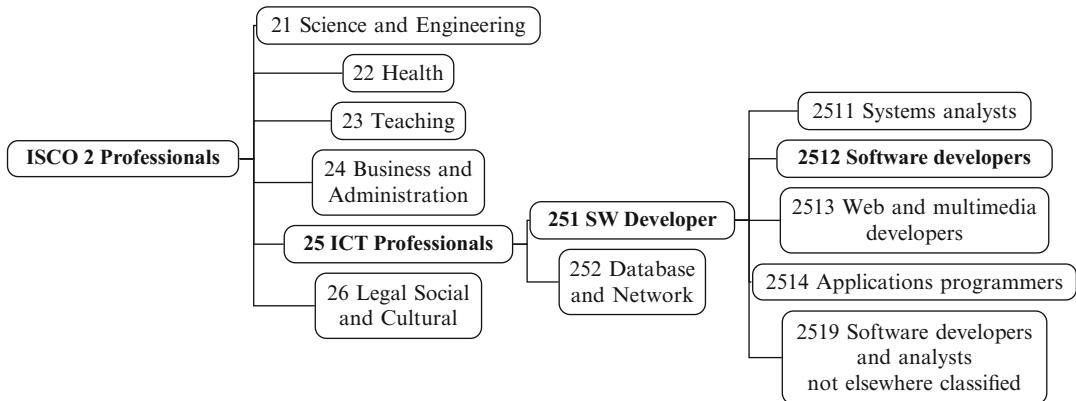
Contract type: Permanent

An example of real Web job vacancies as collected in the

(B) **Application Consultant.** “We are seeking for an application consultant that will support our internal SW engineers in the realization of ad hoc ERP software. The ideal candidate should have (at least) a high-level degree and 2+ years experience in using both Sharepoint-MS CRM and Magic. Coding skills on VB are appreciated.” [ISCO 3512: ICT user support technicians]

Workplace: Rome

Contract type: Unlimited term



Big Data Enables Labor Market Intelligence, Fig. 1 A branch of the ISCO classification tree for professionals

A was classified on the code 2512: *Software Developer* of the ISCO hierarchical classifier by labor market experts of The European Network on Regional Labour Market Monitoring (2017), while job offer B refers to 3512: *ICT user support technicians* (an example is provided in Fig. 1).

In addition, focusing on skills (both hard and soft) have to be identified from raw text and linked against the skill taxonomy (e.g., the ESCO taxonomy), while new emerging skills have to be recognized and returned as well. *Which are the most important in skill in these advertisements? Which are the most relevant skills in these job occupations?* These are just few questions that labor market analysts and specialists need to face to deeply understand the labor market at local, national, and international level and to design labor policies accordingly.

Research Findings

Literature on LMI. Labor market intelligence is an *emerging cross-disciplinary field of studies* that is gaining research interests in both *industrial* and *academic* communities. On one side, the task of extracting useful information from unstructured texts (e.g., vacancies, curricula, etc.) mainly focused on the e-recruitment process, by attempting to support or to automate the *resume* management by matching candidate profiles with job descriptions using language models and ma-

chine learning (Li et al. 2017; Zhu et al. 2016; Kokkodis et al. 2015; Verroios et al. 2015; Singh et al. 2010; Yi et al. 2007; Hong et al. 2013). On the other side, companies need to automatize human resource (HR) department activities; as a consequence, a growing amount of commercial skill-matching products have been developed in the last years, for instance, BurningGlass, Workday, Pluralsight, EmployInsight, and TextKernel (see, e.g., Neculoiu et al. 2016). To date, the only commercial solution that uses international standard taxonomies is Janzz: a Web-based platform to match labor demand and supply in both public and private sectors. It also provides API access to its knowledge base, but it is not aimed at classifying job vacancies. Worth of mentioning is Google Job Search API, a pay-as-you-go service announced in 2016 for classifying job vacancies through the Google Machine Learning service over O*NET, that is the US standard occupation taxonomy. Although this commercial service is still a closed alpha, it is quite promising and also sheds the light on the needs for reasoning with Web job vacancies using a common taxonomy as baseline.

Calls for Projects on LMI. Focusing on *public administrations and organizations*, there is a growing interest in designing and implementing real LMI applications to Web labor market data for supporting the policy design and evaluation activities through evidence-based decision-

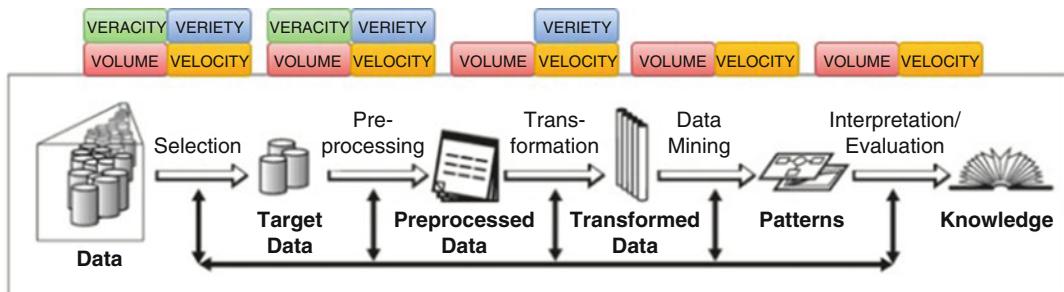
making. In 2010 the European Commission has published the communication “A new impetus for European Cooperation in Vocational Education and Training (VET) to support the Europe 2020 strategy” (CEDEFOP 2010) aimed at promoting education systems, in general, and VET, in particular. In 2016, the European Commission highlighted the importance of vocational and educational activities, as they are “valued for fostering job-specific and transversal skills, facilitating the transition into employment and maintaining and updating the skills of the workforce according to sectorial, regional, and local needs” (European Commission 2016a). In 2016, the EU and Eurostat launched the ESSnet Big Data project (EuroStat 2016), involving 22 EU member states with the aim of “integrating big data in the regular production of official statistics, through pilots exploring the potential of selected Big Data sources and building concrete applications.” Furthermore, in 2014 the EU CEDEFOP agency – aimed at supporting the development of European Vocational Education and Training – has launched a call-for-tender for realizing a system able to collect and classify Web job vacancies from five EU countries (CEDEFOP 2014). The rationale behind the project is to turn data extracted from Web job vacancies into knowledge (and thus value) for policy design and evaluation through a fact-based decision-making. Given the success of the prototype, a further call-for-tender has been opened for realizing the Web labor market monitor for the whole EU, including 28 EU country members and all the 24 languages of the Union (CEDEFOP 2016).

All these initiatives are quite effective and shed the light on the relevance of reasoning with labor market information for supporting business purposes in both private sector (e.g., *job matching*, *candidate profile*, etc.) and in the public one (e.g., *labor market analysis*).

KDD Applied to LMI

The extraction of knowledge from (Big) labor market data has been addressed using the KDD approach as baseline (i.e., knowledge discovery in databases). KDD is mainly composed of five steps, as shown by Fayyad et al. (1996). Clearly, this approach needs to be adapted to the domain of interest, enhancing one task or step in spite of another. To this end, Fig. 2 reports a graphical overview of the KDD framed within the LMI context, along with a number of labels that specify which Vs of the Big Data context is involved. Each step can be adapted to the Web labor market context as follows.

Step 1: Selection. Data sources have to be selected first. Each Web source has to be evaluated and ranked to guarantee the reliability of the information included (see, e.g., the work concerning the *believability* of data by Wang and Strong 1996). Just to give an example, this phase should take into account the vacancy publication date, the website’s update frequency, the presence of structured data, if there are any restrictions in downloading, etc. At the end of this process, a *rank* of reliable Web sources has been produced. As one might note, this step requires to deal with all the 4 Vs of Big Data that include also



Big Data Enables Labor Market Intelligence, Fig. 2 The KDD steps annotated to specify which Vs of Big Data is involved in each step. Figure built on top of the figure from Fayyad et al. (1996)

the *veracity*, intended as the biases, noise, and abnormality in data.

Step 2: Preprocessing. This includes the data cleaning task to remove noise from the data or inappropriate outliers (if any), deciding how to handle missing data, as well as identifying a function to detect and remove duplicated entries (e.g., duplicated vacancies, vacancy with missing values, etc.). Data quality and cleaning tasks are indeed mandatory steps of any data-driven decision-making approach for guaranteeing the believability of the overall process, intended as “the extent to which data is accepted or regarded as true, real, and credible” (see, e.g., Wang and Strong 1996; Redman 1998; Mezzanzanica et al. 2015; Batini et al. 2009).

Notice that the identification of duplicated job vacancies is far from straightforward. Indeed, job vacancies are usually posted on different website, and the *same* text is often reused to advertise a similar position. The former is a duplication while the latter not. The identification of appropriate features to correctly identify duplicates is crucial in the Web labor market domain. Notice that this step allows reducing the complexity of Big Data scenario, as the impact of the *veracity* dimension is mitigated through data quality and cleaning activities performed within this task.

Step 3: Transformation. This step includes the data reduction and projection tasks, which aim at identifying a unified model to represent the data, depending on the goal of the task. Furthermore, it may include the use of dimensionality reduction or transformation methods to reduce the effective number of variables or to find invariant representations for the data. Still in this case, this step reduces the complexity of the dataset by addressing the *variety* dimension. This step is usually performed by means of ETL techniques (extract, transform, and load) that aim at supporting the data preprocessing and transformation tasks in the KDD process. Roughly speaking, in the ETL process, the data extracted from a source system undergoes a set of transformation routines that analyze, manipulate, and then cleanse the data before loading them into a knowledge base (see,

e.g., Vassiliadis et al. 2002 for details on the ETL process). Notice that, at the end of this step, the *variety* issue related to Big Data is expected to be solved as a cleansed and well-defined model of the data is reached.

Step 4: Data Mining and Machine Learning.

This step requires to identify proper AI algorithms (e.g., classification, prediction, regression, clustering), searching for patterns of interest in a particular representational form on the basis of the analysis purposes. More specifically, in the context of LMI, it usually requires the use of text classification algorithms (ontology based, machine learning based, etc.) that builds a classification function to map a data item into one of several predefined classes. Here, items are represented by Web job offers, while the predefined classes are the one from a taxonomy (e.g., a proprietary one or a public one, as ESCO, O*NET, etc.). This is the core step of the KDD approach, even for LMI activity. Therefore, the task of job vacancy classification deserves to be formally described in terms of text categorization.

Text categorization aims at assigning a Boolean value to each pair $(d_j, c_i) \in D \times C$ where D is a set of documents and C a set of predefined categories. A *true* value assigned to (d_j, c_i) indicates document d_j to be set under the category c_i , while a *false* value indicates d_j cannot be assigned under c_i . In the LMI scenario, a set of job vacancies \mathcal{J} can be seen as a collection of documents each of which has to be assigned to one (and only one) occupation code of the taxonomy. Hence, classifying a job vacancy over a classification system requires to assign one occupation code to a job vacancy. This text categorization task can be solved through machine learning, as specified by Sebastiani (2002). Formally speaking, let $\mathcal{J} = \{J_1, \dots, J_n\}$ be a set of job vacancies and the classification of \mathcal{J} under the taxonomy consists of $|O|$ independent problems of classifying each job vacancy $J \in \mathcal{J}$ under a given taxonomy occupation code o_i for $i = 1, \dots, |O|$. Then, a *classifier* is a function $\psi : \mathcal{J} \times O \rightarrow \{0, 1\}$ that approximates an unknown target function $\dot{\psi} : \mathcal{J} \times O \rightarrow \{0, 1\}$.

Clearly, as this case requires to deal with a single-label classifier, $\forall j \in \mathcal{J}$ the following constraint must hold: $\sum_{o \in O} \psi(j, o) = 1$.

Step 5: Interpretation/Evaluation. Finally, this step employs visual paradigms to visually represent the resulting knowledge according to the user's purposes. In the LMI context, this step requires to take into account the user's ability in understanding the data and its main goal in the LMI field. For example, public administrations might be interested in identifying the most requested occupations in its local area; companies might focus on monitoring skills trend and identifying new skills for some occupations, so that they can promptly design training path for their employees, etc. In the last few years, a lot of work has been done for producing off-the-shelf visual libraries that implement several narrative and visual paradigms. A powerful example is D3js (Bostock et al. 2011), a data-driven responsive library for producing dynamic, interactive data visualization even in Big Data context (see, e.g., Bao and Chen 2014).

Examples of Application and Projects

As said above, LMI is an emerging field, in which the private sector plays a key role through many commercial solutions while public and academic applications/projects are fewer. Focusing on the latter, here two EU projects are reported as they are mature examples of how to use Big Data for LMI purposes.

Application 1: LMI for Decision-Making

In 2014, the CEDEFOP EU Agency invited academics to participate a call-for-tender aimed at collecting Web job vacancies from five EU countries and extracting the requested skills from the data. The rationale behind the project is to turn data extracted from Web job vacancies into knowledge (and thus value) for policy design and evaluation through fact-based decision-making. The system (Boselli et al. 2017a) is now running on the CEDEFOP data center since June 2016, gathering and classifying job vacancies

from five EU countries, namely, the United Kingdom, Ireland, Czech Republic, Italy, and Germany, through machine learning (Boselli et al. 2017b). To date, the system collected 7+ million job vacancies over the five EU countries, and it accounts among the research projects that a selection of Italian universities addressed in the context of Big Data (Bergamaschi et al. 2016).

This prototype allowed the CEDEFOP agency to have evidence of the *competitive advantage* that the analysis of Web job vacancy enables with respect to the classical survey-based analyses, in terms of (i) near-real-time labor market information; (ii) reduced time-to-market (iii) fine-grained territorial analysis, from countries down to municipalities; (iv) identification of the most requested skills; and (v) evaluation of skills impact to each ISCO profession.

Online Demo. To better clarify the final outcome of a LMI project, the reader can refer to the animated heat map showing the job vacancy collection progress over time available at <http://goo.gl/MQuIUn>. On the other side, an example of how the LMI knowledge might be made available to decision-maker is provided at <http://goo.gl/RRb63S>.

Application 2: LMI for Producing Official Statistics

Focusing on the statistical perspective related to the analysis of Big Data for LMI, the Eurostat – the official statistical office of the European Union – launched a project named ESSnet Big Data aimed at integrating Big Data about LMI in the regular production of official statistics, through pilots exploring the potential of selected Big Data sources and building concrete applications (EuroStat 2016). The project – started in late 2016 – is composed of 22 EU partners and strictly follows the KDD approach to collect data from Web portals previously ranked and cleanse and transform it for classifying data over standard taxonomies. Worth noting here is the important role that the *representativeness* of Big Data plays in this project, as participants aim at evaluating the ability of Big Data to be representative of the whole population (or a stratified sample) for being included in official EU statistics.

Future Directions for Research

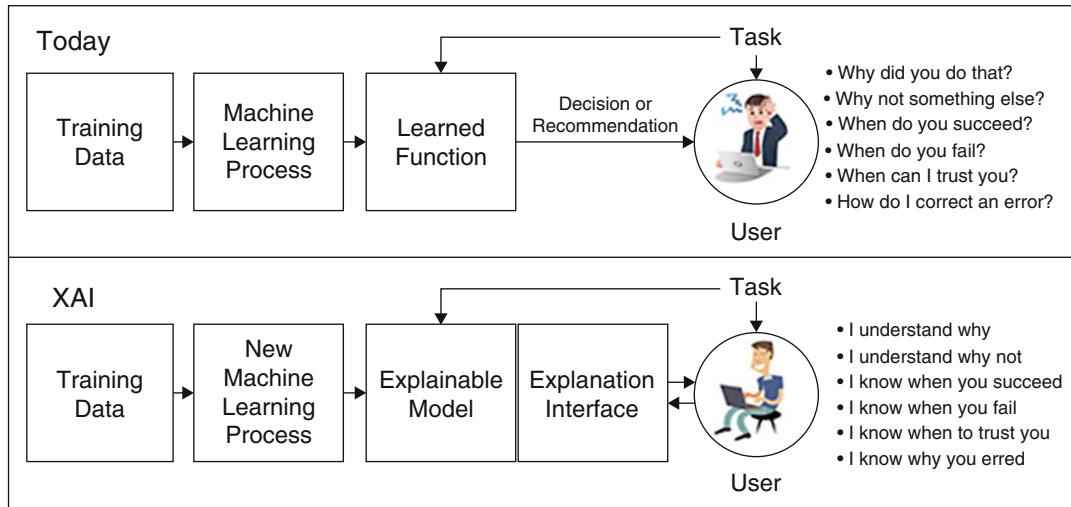
LMI is a cross-disciplinary research area that involves computer scientists, statisticians, and economists working together to make sense of labor market data for decision-making. In such a scenario, three distinct research directions can be identified for each area of interest, as discussed below.

K1: Making LMI Explainable. Explainable artificial intelligence (XAI) is an emerging branch of artificial intelligence aimed at investigating new machine learning models able to be completely explainable in contexts for which transparency is important, for instance, where these models are adopted to solve analysis (e.g., classification) or synthesis tasks (e.g., planning, design), as well as mission critical applications. DARPA recently launched the Explainable AI (XAI) program that aims to create a suite of AI systems able to explain their own behavior, focusing on ML and deep learning techniques. To date, indeed, most ML research models rarely provide explanations/justifications for the outcomes they offer in the tasks they are applied to. The need for explainable AI indeed is motivated mainly by the need for (i) trust, (ii) interaction, and (iii) transparency, as recently discussed in by Fox et al. (2017). This is also the reason behind the interest in XAI applications by academia and industrial communities (see, e.g., Biran and Cotton 2017; Ribeiro et al. 2016). Focusing on business contexts, as LMI is, autonomous decision-making can be framed as a set of economic problems that need to make evidence of the rationale behind the action suggested, so that the final decision can result believable to human decision-makers. Figure 3 would give a graphical overview of the main goal of XAI, that is, the development of AI-based systems able to explain their rationale, to characterize their strengths and weaknesses, as well as to convey an understand how they will behave in the future.

XAI in LMI. In the LMI field, indeed, ML-based approaches do not provide any explanations for their outcomes/predictions. This untransparent behavior may prevent the decision-

maker to consider reliable enough the analyses that have been computed, making the overall process ineffective. In essence, the use of XAI algorithms in LMI fits this challenging issue that DARPA identifies working on *machine learning problems to construct decision policies for an autonomous system to perform a variety of simulated missions* (DARPA 2017). Specifically, providing explanation of a job vacancy classifier would request to answer a question as *why a job vacancy J has been classified as financial manager rather than financial and insurance manager?*

To better understand what an explanation should be, Fig. 4 reports an example where the LIME tool proposed by Ribeiro et al. (2016) has been applied to retrieve the explanation of the example above, explaining a classifier that was previously trained on million English job vacancies. As one might note, the system provides the probabilities that lead the ML algorithm to classify the job vacancy on a given ISCO code. Furthermore, it also provides the contribution that each feature (here, word) has given to the classification process. We can observe that terms *finance* and *manager* positively contributed in classifying on financial manager (code 1211). Surprisingly, the system also reports that the term *finance* has a higher probability to classify a job vacancy on financial and insurance manager (code 1346), but the presence of term *manager* here contributed negatively in classifying on that code. Such a kind of result is quite useful for improving both transparency and interaction of ML systems in LMI applications. In essence, this real example sheds the light on what happens behind the scenes of a black-box classifier: Although the system correctly classifies the vacancy on the right code, the negative contribution that the term “*manager*” gives to the class with label 1346 is wrong and might have unpredictable effects on the classification in the future. On the other side, having such a kind of information would allow analyst to revise and refine the system, applying a sort of *explain and validate* approach to improve both the effectiveness and reliability of the machine learning-based system.



Big Data Enables Labor Market Intelligence, Fig. 3 Graphical overview of a XAI system (Taken from www.darpa.mil)



Big Data Enables Labor Market Intelligence, Fig. 4
An example of explanation for a ML-based system. ESCO codes are as follows. 1211,“financial manager”;

1346, “financial and insurance services branch managers”; 3313,“accounting associate professionals”; 2413, “financial analysts”

This, in turn, would allow final users to answer the following questions:

- Q1: Why did you classify J on that code?
- Q2: Which feature has lead you to classify J on code A rather than B?
- Q3: Which is the confidence score in your classification outcome?
- Q4: How robust your classification is to errors and changes in labor market?

Answering to these questions in a believable manner is a crucial step to make the decision-maker in state to *take reliable decisions and design effective policies* when the data is processed and managed by AI-based systems.

K2: Dealing with cross-language LMI. The problem of dealing with multilingual Web job

vacancy is growing in importance, mainly in the context of LMI for decision-making, where one of the main goal is to be able to share and compare labor market dynamics among different countries.

- Q1: How to compare labor market for different countries?
- Q2: How to deal with jargon and lexicon of different countries?
- Q3: How to deal with different labor market taxonomy of each country?

To this end, there is call by European Community in using the ESCO standard taxonomy that would provide a *lingua franca* to express occupations, skills, and qualifications over 28 official EU languages. Unfortunately, the use of ESCO does

not completely solve the issue related to multi-language job vacancies, as a *training/test set* has still to be created for each language. To cope with this issue, the community working on LMI is working on two distinct directions:

- Build a classifier for a text classification task in some target language T , given labeled training tests for the same task in a different source language S . Clearly, a function that maps word correspondence between languages T and S is required. This procedure is called *cross-language text classification*; see, e.g., Shi et al. (2010), Bel et al. (2003), and Olsson et al. (2005);
- Use unlabeled data from both languages and a word translation map to *induce* cross-lingual word correspondences. In this way, a cross-lingual representation is created enabling the transfer of classification knowledge from the source to the target language, aka *cross-language structural correspondence learning*; see, e.g., Prettenhofer and Stein (2011).

K3: Representativeness of Web Labor Market Information.

A key issue related to the use of Big Data for decision-making relies on their ability of being representative of the whole population. Indeed, while on one side labor market surveys can be designed for being representative of a certain population (sectors/occupations etc.), on the other side, representativeness of web data might not be evaluated easily, and this affects the reliability of statistics and analytics derived from it. This issue also affects the labor market intelligence field (e.g., some occupations, and sectors are not present in web advertisements). To cope with this issue, the question has been rephrased to assess whether a set of online job vacancies is a representative sample of *all* job vacancies *in a specified economic context*. Indeed, even though it should be considered finite, the population of job vacancies continuously changes, and this makes difficult to exactly determine it in stable manner. Indeed, jobs and people are usually reallocated in the labor market and in firms, respectively; tasks are split, restructured,

or partially switched, and recruitment strategies might have sectoral and occupational specificities (see, e.g., Gosling et al. 2004; Mang 2012). From a statistical perspective, samples of available Web vacancies are prone to problems of self-selection and/or non-ignorable missing mechanism (Little and Rubin 2014).

In such a scenario, several approaches have been taken by different authors in attempting to deal with representativeness issues related to the usage of online job vacancy data for analytical and policy-making purposes. Among others, a number of studies decided to focus on the segment of the labor market characterized by widespread access to the Internet (e.g., ICT) in the belief that they would be relatively well represented among the job applicants due to characteristics of Internet users and would therefore offer a representative data sample (Anne Kennan et al. 2006; Wade and Parent 2002). Notice that this approach has been considered as valid by EU commission and that in 2016 invited to contribute to a call-for-tender aimed at *obtaining data on job vacancies for ICT practitioners and other professional categories gathered by crawling job advertisements published in relevant online job advertisement outlets* (European Commission 2016b).

Recently, some researchers considered online data generalizable due to the dominant market share and very high reputation of the chosen portal among employers and employees (Kureková et al. 2012).

Nonetheless, the problem related to representativeness of Big Data concerning LMI data – that is important for including them in official statistics and data analytics – still remains an open issue.

Concluding Remarks

LMI is an emerging and cross-disciplinary field of study that draws competences from different disciplines, such as of computer science, statistics, and economics in order to make sense of Big Data in the labor market field. Though the rationale behind LMI is the *use* of labor market

data, the final goal that guides LMI activities may vary as the purpose of the users varies. Private companies aim at supporting the HR in recruitment activities, while analysts and public organizations would use LMI for studying the labor market dynamics and for designing more effective policies following a data-driven approach. In any case, all the approaches mentioned strictly rely on the KDD approach as baseline and massively employ AI algorithms for classifying occupations/resumes and extract occupations and skills. As for the AI in general, in the early future, the research on LMI will overcome some limitations that might prevent a steady use of Big Data in the LMI field, as the ability to *explain* and motivate to decision-makers how AI works behind the hood; the ability to deal with multiple languages, lexicon, and idioms with a limited effort; and the ability to be representative – or at least statistically relevant – of the whole population observed. These are the research directions identified by top players on LMI – mainly EU and US governments – for allowing Big Data to enable labor market intelligence.

Cross-References

- [Big Data Visualization Tools](#)
- [Data Cleaning](#)
- [ETL](#)

References

- Anne Kennan M, Cole F, Willard P, Wilson C, Marion L (2006) Changing workplace demands: what job Ads tell us. In: Aslib proceedings, vol 58. Emerald group publishing limited, pp 179–196
- Bao F, Chen J (2014) Visual framework for big data in d3.js. In: 2014 IEEE workshop on electronics, computer and applications. IEEE, pp 47–50
- Batini C, Cappiello C, Francalanci C, Maurino A (2009) Methodologies for data quality assessment and improvement. ACM Comput Surv (CSUR) 41(3):16
- Bel N, Koster CH, Villegas M (2003) Cross-lingual text categorization. In: ECDL, vol 2769. Springer, pp 126–139
- Bergamaschi S, Carlini E, Ceci M, Furletti B, Giannotti F, Malerba D, Mezzanzanica M, Monreale A, Pasi G, Pedreschi D et al (2016) Big data research in Italy: a perspective. Engineering 2(2): 163–170
- Biran O, Cotton C (2017) Explanation and justification in machine learning: a survey. In: IJCAI-17 workshop on explainable AI (XAI), p 8
- Boselli R, Cesaroni M, Marrara S, Mercurio F, Mezzanzanica M, Pasi G, Viviani M (2017a) Wolmis: a labor market intelligence system for classifying web job vacancies. J Intell Inf Syst, 1–26. <https://doi.org/10.1007/s10844-017-0488-x>
- Boselli R, Cesaroni M, Mercurio F, Mezzanzanica M (2017b) Using machine learning for labour market intelligence. In: The European conference on machine learning and principles and practice of knowledge discovery in databases – ECML-PKDD
- Bostock M, Ogievetsky V, Heer J (2011) D³ data-driven documents. IEEE Trans Vis Comput Graph 17(12):2301–2309
- CEDEFOP (2010) A new impetus for European cooperation in vocational education and training to support the Europe 2020 strategy (publicly available at <https://goo.gl/Goluxo>)
- CEDEFOP (2014) Real-time labour market information on skill requirements: feasibility study and working prototype. CEDEFOP reference number ao/rpa/vkvet-nsofro/real-time lmi/010/14. Contract notice 2014/s 141-252026 of 15 July 2014. <https://goo.gl/qNjmrn>
- CEDEFOP (2016) Real-time labour market information on skill requirements: setting up the EU system for online vacancy analysis ao/dsl/vkvet-grusso/real-time lmi 2/009/16. contract notice – 2016/s 134-240996 of 14 July 2016. <https://goo.gl/5FZS3E>
- DARPA (2017) Explainable artificial intelligence (XAI). Available at <https://www.darpa.mil/program/explainable-artificial-intelligence>
- European Commission (2016a) A new skills agenda for Europe. COM(2016) 381/2
- European Commission (2016b) Vacancies for ICT online repository 2 (victory 2): data collection. <https://goo.gl/3D7qNQ>
- European Commission (2017) ESCO: European skills, competences, qualifications and occupations. Available at <https://ec.europa.eu/esco/portal/browse>
- EuroStat (2016) The ESSnet big data project. Available at <https://goo.gl/EF6GuU>
- Fayyad U, Piatetsky-Shapiro G, Smyth P (1996) The KDD process for extracting useful knowledge from volumes of data. Commun ACM 39(11):27–34
- Fox M, Long D, Magazzeni D (2017) Explainable planning. arXiv preprint arXiv:170910256
- Frey CB, Osborne MA (2017) The future of employment: how susceptible are jobs to computerisation? Technol Forecast Soc Chang 114(Supplement C):254–280. <https://doi.org/10.1016/j.techfore.2016.08.019>, <http://www.sciencedirect.com/science/article/pii/S0040162516302244>
- Gosling SD, Vazire S, Srivastava S, John OP (2004) Should we trust web-based studies? A comparative analysis of six preconceptions about internet questionnaires. Am Psychol 59(2):93

- Hong W, Zheng S, Wang H (2013) Dynamic user profile-based job recommender system. In: Computer science & education (ICCSE). IEEE
- Kokkodis M, Papadimitriou P, Ipeirotis PG (2015) Hiring behavior models for online labor markets. In: Proceedings of the eighth ACM international conference on web search and data mining. ACM, pp 223–232
- Kureková L, Beblavý M, Haita C (2012) Qualifications or soft skills? Studying demand for low-skilled from job advertisements. Technical report, NEUJOBS working paper
- Li H, Ge Y, Zhu H, Xiong H, Zhao H (2017) Prospecting the career development of talents: a survival analysis perspective. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 917–925
- Little RJ, Rubin DB (2014) Statistical analysis with missing data. Wiley, New York, USA
- Mang C (2012) Online job search and matching quality. Technical report, IFO working paper
- Mezzananza M, Boselli R, Cesarini M, Mercorio F (2015) A model-based evaluation of data quality activities in KDD. Inf Process Manag 51(2):144–166
- Neculoiu P, Versteegh M, Rotaru M, Amsterdam TB (2016) Learning text similarity with siamese recurrent networks. ACL 2016, p 148
- Olsson JS, Oard DW, Hajic J (2005) Cross-language text classification. In: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval. ACM, pp 645–646
- Organization IL (2017) ISCO: the international standard classification of occupations. Available at <http://www.ilo.org/public/english/bureau/stat/isco/>
- Prettenhofer P, Stein B (2011) Cross-lingual adaptation using structural correspondence learning. ACM Trans Intell Syst Tech (TIST) 3(1):13
- Redman TC (1998) The impact of poor data quality on the typical enterprise. Commun ACM 41(2):79–82
- Ribeiro MT, Singh S, Guestrin C (2016) Why should I trust you?: explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1135–1144
- Sebastiani F (2002) Machine learning in automated text categorization. ACM compu surv (CSUR) 34(1):1–47
- Shi L, Mihalcea R, Tian M (2010) Cross language text classification by model translation and semi-supervised learning. In: Proceedings of the 2010 conference on empirical methods in natural language processing. Association for Computational Linguistics, pp 1057–1067
- Singh A, Rose C, Visweswariah K, Chenthamarakshan V, Kambhatla N (2010) Prospect: a system for screening candidates for recruitment. In: Proceedings of the 19th ACM international conference on information and knowledge management. ACM, pp 659–668
- The European Network on Regional Labour Market Monitoring (2017) Web page (<http://www.regionallabourmarketmonitoring.net>)
- UK Commission for Employment and Skills (2015) The importance of LMI. Available at <https://goo.gl/TcRwvS>
- UK Department for Education and Skills (2004) LMI Matters!
- U.S. Department of Labor/Employment & Training Administration (2017) O*net: the occupational information network. Available at <https://www.onetcenter.org>
- Vassiliadis P, Simitsis A, Skiadopoulos S (2002) Conceptual modeling for ETL processes. In: Proceedings of the 5th ACM international workshop on data warehousing and OLAP. ACM, pp 14–21
- Veroios V, Papadimitriou P, Johari R, Garcia-Molina H (2015) Client clustering for hiring modeling in work marketplaces. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 2187–2196
- Wade MR, Parent M (2002) Relationships between job skills and performance: a study of webmasters. J Manage Inf Syst 18(3):71–96
- Wang RY, Strong DM (1996) Beyond accuracy: what data quality means to data consumers. J Manage Inf Syst 12(4):5–33
- Yi X, Allan J, Croft WB (2007) Matching resumes and jobs based on relevance models. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval. ACM, pp 809–810
- Zhu C, Zhu H, Xiong H, Ding P, Xie F (2016) Recruitment market trend analysis with sequential latent variable models. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 383–392

Big Data for Cybersecurity

Markus Wurzenberger, Florian Skopik, and Giuseppe Settanni
Center for Digital Safety and Security, AIT Austrian Institute of Technology, Vienna, Austria

Synonyms

[Computer security](#); [IT security](#)

Definitions

Cybersecurity deals with the protection of computer systems; information and communication technology (ICT) networks, such as enterprise

networks, cyber-physical systems (CPS), and the Internet of Things (IoT); and their components, against threats that aim at harming, disabling, and destroying their software and hardware.

Overview

In the beginning of the Digital Revolution, the primary goal of computer hackers that attack ICT infrastructures and networks was just to receive recognition from like-minded people. Thus, the consequences have been mostly downtimes and the often costly need of recovering and cleaning up compromised systems. Therefore, the consequences were manageable with rather little effort. In the meantime, the ICT networks' relevance massively increased, and they became the vital backbone of economic as well as daily life. Since their importance and complexity continuously evolve, also cyberattacks against them become more complex and diverse and consequently difficult to detect. Today's modern attacks usually target on stealing intellectual properties, sabotaging systems, and demanding ransom money. Sophisticated and tailored attacks are called advanced persistent threats (APT) (Tankard 2011) and usually result in high financial loss and tarnished reputation.

Since usually the protection against sophisticated cyberattacks is difficult and highly challenging, (i) timely detection to limit the caused damage to a minimum, which can be achieved by applying smart *intrusion detection systems (IDS)*, and (ii) preventive actions, such as raising situational awareness, through analyzing and sharing *cyber threat intelligence (CTI)*, are of primary importance. Because of the continuously growing interconnection of digital devices, the amount of generated data and new malware samples per day grows with an exorbitant speed. Moreover, particular types of malware, such as worms, rapidly evolve over time and spread over the network in unpredictable ways (Dainotti et al. 2007; Kim et al. 2004). Thus, it gets more and more difficult to analyze all generated data that might include hints on cyberattacks and malicious system behavior. This is the reason why cybersecurity be-

came a *big data problem*. According to the four Vs feature of big data (Chen et al. 2014), big data is defined by (i) volume (large amount of data), (ii) variety (heterogeneous and manifold data), (iii) velocity (rapidly generated data that has to be analyzed online, i.e., when the data is generated), and (iv) value (extraction of huge value from a large amount of data).

This section deals with the two main research areas of cybersecurity that are related to big data: (i) intrusion detection, which deals with revealing cyberattacks from network traces and log data, and (ii) CTI analysis and sharing, which supports incident response and decision-making processes. Therefore, the section first splits into two main parts, which are structured the same way. Both first answer the question why the considered topic is a big data problem and then summarize the state of the art. Finally, future directions for the research of big data for cybersecurity are presented. Thus, it is also discussed how intrusion detection and CTI analysis and sharing can benefit from each other.

Intrusion Detection

Countermeasures and defense mechanisms against cyberattacks split into tools that aim at timely detection of attack traces and tools that try to prevent attacks or deploy countermeasures in time. Technical preventive security solutions are, for example, firewalls, antivirus scanners and digital rights, account management, and network segmentation. Tools that aim at detecting cyberattacks and therefore only monitor a network are called intrusion detection systems (IDS). Currently applied IDS mostly implement signature-based black-listing approaches. Black-listing approaches prohibit system behavior and activities that are defined malicious. Hence, those solutions can only detect already known attack patterns. Nevertheless these established tools build the vital backbone of an effective security architecture and ensure prevention and detection of the majority of threats. Nevertheless, the wide range of application possibilities of modern devices and the fast changing threat landscape

demand smart, flexible, and adaptive white-listing-based intrusion detection approaches, such as self-learning anomaly-based IDS. In contrast to black-listing approaches, white-listing approaches permit a baseline of normal system behavior and consider every deviation of this ground truth as malicious activity. Thus, it is possible to detect previously unknown attacks.

Intrusion Detection: A Big Data Problem

Because of the rapidly growing number of interconnected digital devices, also the amount of generated data increases. Thus, the *volume* of data that potentially can be analyzed for intrusion detection is enormous. Hence, approaches are required that can analyze and also correlate a huge load of data points. Furthermore, there is a wide *variety* of data types available for intrusion detection. The spectrum ranges from binary files over network traffic to log data, just to name a few. Additionally the representation of this data differs depending on the applied technologies, implementations, and used services, as well as operating systems, why analysis of data created in an ICT network comprising various different types of devices is not straightforward. Furthermore, the data is generated quickly, and the *velocity* is increasing with the number of connected devices. Thus, to provide online intrusion detection, i.e., the data is analyzed when it is generated, which ensures timely detection of attacks and intruders, big data approaches are required that can process rapidly large amounts of data. Since most of the generated data relates to normal system behavior and network communication, it is important to efficiently filter the noise, so only relevant data points that include information of high *value* for intrusion detection remain.

Intrusion Detection: Methods and Techniques

Like other security tools, IDS aim to achieve a higher level of security in ICT networks. Their primary goal is to timely detect cyberattacks, so that it is possible to react quickly and therefore reduce the window of opportunity of the attacker to cause damage. IDS are passive security mechanisms, which only detect attacks without

setting any counter measurements automatically (Whitman and Mattord 2012).

In 1980, James Anderson was one of the first researchers who indicated the need of IDS and contradicted the common assumption of software developers and administrators that their computer systems are running in “friendly,” i.e., secure, environments (James P Anderson 1980). Since the 1980s, the amount of data generated in ICT networks tremendously increased, and intrusion detection therefore became a big data problem. Furthermore, since Anderson published his report on IDS, a lot of research in this area has been done until today (Axelsson 2000; Sabahi and Movaghfar 2008; Liao et al. 2013; Garcia-Teodoro et al. 2009).

IDS can be roughly categorized as host-based IDS (HIDS), network-based IDS (NIDS), and hybrid or cross-layer IDS that make use of both data collected on hosts and monitored network traffic (Vacca 2013; Sabahi and Movaghfar 2008). There exist generally three methods that are used in IDS: (i) signature-based detection (SD), (ii) stateful protocol analysis (SPA), and (iii) anomaly-based detection (AD) (Liao et al. 2013).

- **SD (knowledge based)** uses predefined signatures and patterns to detect attackers. This method is simple and effective for detecting known attacks. The drawback of this method is it is ineffective against unknown attacks or unknown variants of an attack, which allows attackers to evade IDS based on SD. Furthermore, since the attack landscape is rapidly changing, it is difficult to keep the signatures up to date, and thus maintenance is time-consuming (Whitman and Mattord 2012).
- **SPA (specification based)** uses predetermined profiles that define benign protocol activity. Occurring events are compared against these profiles, to decide if protocols are used in the correct way. IDS based on SPA track the state of network, transport, and application protocols. They use vendor-developed profiles and therefore rely on their support (Scarfone and Mell 2007).

- **AD (behavior based)** approaches learn a baseline of normal system behavior, a so-called ground truth. Against this ground truth, all occurring events are compared to detect anomalous system behavior. In opposite to SD- and SPA-based IDS, AD-based approaches allow detection of previously unknown attacks. A drawback of AD-based IDS is the usually high false-positive rate (Garcia-Teodoro et al. 2009; Chandola et al. 2009).

Modern sophisticated and tailored attacks, such as APTs, where attackers try to hide their presence as long as possible and therefore try to circumvent detection by signatures and blacklists and respect benign protocol activity, require flexible approaches that provide broader security. A solution to that are white-listing approaches that make use of a known baseline of normal system/network behavior. Furthermore, it is impossible to define a complete set of signatures, because during normal usage, a network reaches only a small number of possible states, while the larger number remains unknown. Thus, while in case of white-listing, an incomplete baseline of normal system/network behavior would cause a larger number of false alarms, i.e., a bigger number of alarms that have to be further analyzed, and an incomplete blacklist leads to false negatives, i.e., undetected attacks. Thus, flexible AD approaches are required that are able to detect novel and previously unknown attacks. While SD and SPA solutions are usually easier to deploy than AD, they depend on the support of vendors. Hence, they mostly cannot be applied in legacy systems and systems with small market shares, which are often poorly documented and not supported by vendors, but often used in industry networks, such as industrial control systems and IoT.

The rapidly changing cyber threat landscape demands flexible and self-adaptive IDS approaches. Since networks grow fast and therefore also the amount of generated data, this task requires machine learning-based big data approaches that implement self-learning

AD-based IDS. These self-learning solutions usually learn during a training phase, or after they are deployed, a baseline of normal system/network behavior that then serves as ground truth to detect anomalies that expose attacks and especially invaders. Generally, there are three ways how self-learning AD can be realized (Goldstein and Uchida 2016):

1. **Unsupervised:** This method does not require any labeled data and is able to learn to distinguish normal from malicious system behavior while it is deployed without any training. It classifies any monitored data as normal or anomalous.
2. **Semi-supervised:** This method is applied when the training set only contains anomaly-free data and is therefore also called “one-class” classification.
3. **Supervised:** This method requires a fully labeled training set containing both normal and malicious data.

These three methods do not require active human intervention during the learning process. While unsupervised self-learning is entirely independent from human influence, for the other two methods, the user has to ensure that the training data is anomaly-free or correctly labeled. To implement self-learning AD, there are many machine learning algorithms (Witten et al. 2016) that can be applied for that big data task. Methods that are used for machine learning in cybersecurity are, for example (Buczak and Guven 2016):

- **Artificial neural networks (ANN):** Input data activates neurons (nodes) of an artificial network, inspired by the human brain. The nodes of the first layer pass their output to the nodes of the next layer, until the output of the last layer of the artificial network classifies the monitored ICT networks’ current state (Cannady 1998).
- **Bayesian networks:** Bayesian networks define graphical models that encode the probabilistic relationships between variables of interest and can predict consequences of

actions (Heckerman et al. 1998).

- **Clustering:** Clustering enables grouping of unlabeled data and is often applied to detect outliers (Xu and Wunsch 2005).
- **Decision trees:** Decision trees have a treelike structure, which comprises paths that lead to a classification based on the values of different features (Safavian and Landgrebe 1991).
- **Hidden Markov models (HMM):** A Markov chain connects states through transition probabilities. HMM aim at determining hidden (unobservable) parameters from observed parameters (Baum and Eagon 1967).
- **Support vector machines (SVM):** SVM construct hyperplanes in a high- or infinite-dimensional space, which then can be used for classification and regression. Thus, similar to clustering, SVM can, for example, be applied for outlier detection (Steinwart and Christmann 2008).

Cyber Threat Intelligence

The concepts of *data*, *information*, and *intelligence* differ from one another in terms of volume and usability. While data is typically available in large volumes, and describes individual and unarguable facts, information is produced when a series of data points are combined to answer a simple question. Although information is a far more useful output than the raw data, it still does not directly inform a specific action. Intelligence takes this process a stage further by interrogating data and information to tell a story (e.g., a forecast) that can be used to inform decision-making. Crucially, intelligence never answers a simple question; rather, it paints a picture that can be used to help people answer much more complicated questions. Progressing along the path from data to information to intelligence, the quantity of outputs drops off dramatically, while the value of those outputs rises exponentially (Chairman of the Joint Chiefs of Staff 2013).

Modern cybersecurity incidents manifest themselves in different forms depending on the attack perpetrators' goal, on the sophistication of the employed attack vectors, and on the

information system the attack targets. Large amounts of data are to be analyzed while handling such incidents in order to derive meaningful information on the relations existing among the collected data units, and eventually obtain cyber threat intelligence (CTI), with the purpose of designing suitable reaction strategies and timely mitigate the incident's effects.

CTI: A Big Data Problem

Hundreds of thousands of new malware samples are created every day. Considering the large amount of vulnerabilities and exploits revealed at a similar rate, there exists a large *volume* of data comprising information on cyber threats and attacks that can be analyzed to improve incident response, to facilitate the decision-making process, and to allow an effective and efficient configuration of cybersecurity mechanisms. Furthermore, the *variety* of the data is manifold. There exist, for example, many different standards describing vulnerabilities, threats, and incidents that follow different structures, as well as threat reports and fixes written in free text form. Because of the large amount of data on cyberattacks generated daily, the data that can be analyzed to derive intelligence is created with a high *velocity* and should be processed as fast as possible. Since large parts of the data are just available as free text and computer networks show a high diversity among each other, it is of prime importance to select and extract the information with the highest *value*.

From Raw Data to Cyber Threat Intelligence

Threat information may originate from a wide variety of internal and external data sources. Internal sources include security sensors (e.g., intrusion detection systems, antivirus scanners, malware scanners), logging data (from hosts, servers, and network equipment such as firewalls), tools (e.g., network diagnostics, forensics toolkits, vulnerability scanners), security management solutions (security information and event management systems (SIEMs), incident management ticketing systems), and additionally also personnel, who reports suspicious behavior,

social engineering attempts, and the like. Typical external sources (meaning external to an organization) may include sharing communities (open public or closed ones), governmental sources (such as national CERTs or national cybersecurity centers), sector peers and business partners (for instance, via sector-specific information sharing and analysis centers (ISACs)), vendor alerts and advisories, and commercial threat intelligence services.

Stemming from this variety of sources, it becomes evident that cyber threat intelligence can be (preferably automatically) extracted from numerous technical artifacts that are produced during regular IT operations in organizations:

- **Operating system** services application logs and provides insights into deviations from normal operations within the organizational boundaries.
- **Router, Wi-Fi, and remote services logs** provide insights into failed login attempts and potentially malicious scanning actions.
- **System and application configuration settings and states**, often at least partly reflected by configuration management databases (CMDBs), help to identify weak spots due to not required but running services, weak account credentials, or wrong patch levels.
- **Firewall, IDS, and antivirus logs and alerts** point to probable causes, however, often with high false-positive rates that need to be verified.
- **Web browser histories, cookies, and caches** are viable resources for forensic actions after something happened, to discover the root cause of a problem (e.g., the initial drive-by download).
- **Security information and event management systems (SIEMs)** already provide correlated insights across machines and systems.
- **E-mail histories** are essential sources to learn about and eventually counter (spear) phishing attempts and followed links to malicious sites.
- **Help desk ticketing systems, incident management/tracking systems, and people** provide insights into any suspicious events and

actions reported by humans rather than software sensors.

- **Forensic toolkits and sandboxing** are vital means to safely analyze the behavior of untrusted programs without exposing a real corporate environment to any threat.

B

There are multiple types of potentially useful information obtained from the data sources mentioned before for security defense purposes. Every type has its own characteristics regarding the objective (e.g., to facilitate detection, to support prosecution, etc.), applicability, criticality, and “shareability” (i.e., the effort required to make an artifact shareable because of steps required to extract, validate, formulate, and anonymize some piece of information). Depending on their content, cyber threat information can be grouped into the categories defined by NIST (2016) and OASIS (2017) and described below:

- **Indicators:** are technical artifacts or observables that suggest an attack is imminent or is currently underway or that a compromise may have already occurred. Examples are IP addresses, domain names, file names and sizes, process names, hashes of file contents and process memory dumps, service names, and altered configuration parameters.
- **Tactics, techniques, and procedures (TTPs):** characterize the behavior of an actor. A tactic is the highest-level description of this behavior, while techniques give a more detailed description of behavior in the context of a tactic, and procedures are an even lower-level, highly detailed description in the context of a technique. Some typical examples include the usage of spear phishing emails, social engineering techniques, websites for drive-by attacks, exploitation of operating systems and/or application vulnerabilities, the intentional distribution of manipulated USB sticks, and various obfuscation techniques. From these TTPs, organizations are able to learn how malicious attackers work and derive higher-level and generally valid detection, as well as remediation techniques, compared to

- quite specific measures based on just, often temporarily valid, indicators.
- **Threat actors:** contain information regarding the individual or a group posing a threat. For example, information may include the affiliation (such as a hacker collective or a nation-state's secret service), identity, motivation, relationships to other threat actors, and even their capabilities (via links to TTPs). This information is being used to better understand why a system might be attacked and work out more targeted and effective countermeasures. Furthermore, this type of information can be applied to collect evidences of an attack to be used in court.
 - **Vulnerabilities:** are software flaws that can be used by a threat actor to gain access to a system or network. Vulnerability information may include its potential impact, technical details, its exploitability, and the availability of an exploit, affected systems, platforms, and version, as well as mitigation strategies. A common schema to rate the seriousness of a vulnerability is the common vulnerability scoring schema (CVSS) (Scarfone and Mell 2009), which considers the enumerated details to derive a comparable metric. There are numerous web platforms that maintain lists of vulnerabilities, such as the common vulnerability and exposure (CVE) database from MITRE (<https://cve.mitre.org/>) and the national vulnerability database (NVD) (<https://nvd.nist.gov/>). It is important to notice that the impact of vulnerabilities usually needs to be interpreted for each organization (and even each system), individually, depending on the criticality of the affected systems for the main business processes.
 - **Cybersecurity best practices:** include commonly used cybersecurity methods that have demonstrated effectiveness in addressing classes of cyber threats. Some examples are response actions (e.g., patch, configuration change), recovery operations, detection strategies, and protective measures. National authorities, CERTs, and large industries frequently publish best practices to help organizations building up an effective cyber defense and rely on proven plans and measures.
 - **Courses of actions (CoAs):** are recommended actions that help to reduce the impact of a threat. In contrast to best practices, CoAs are very specific and shaped to a particular cyber issue. Usually CoAs span the whole incident response cycle starting with detection (e.g., add or modify an IDS signature), containment (e.g., block network traffic to command and control server), recovery (e.g., restore base system image), and protection from similar events in the future (e.g., implement multifactor authentication).
 - **Tools and analysis techniques:** this category is closely related to best practices, however focuses more on tools rather than procedures. Within a community it is desirable to align used tools to each other to increase compatibility, which makes it easier to import/export certain types of data (e.g., IDS rules). Usually there are sets of recommended tools (e.g., log extraction/parsing/analysis, editor), useful tool configurations (e.g., capture filter for network protocol analyzer), signatures (e.g., custom or tuned signatures), extensions (e.g., connectors or modules), code (e.g., algorithms, analysis libraries), and visualization techniques.
- Interpreting, contextualizing, and correlating information from different categories allow the comprehension of the current security situation and provide therefore threat intelligence (Skopik 2017). Independently from the type, there are common desired characteristics to make cyber threat intelligence applicable. Specifically, CTI should be obtained *timely*, to allow security operation centers to promptly act; it should be *relevant* in order to be applicable to the pertaining operational environment; it should be *accurate* (i.e., correct, complete, and unambiguous) and *specific*, providing sufficient level of detail and context; finally, CTI is required to be *actionable* to provide or suggest effective course of action.

Future Directions for Research

The following section discusses the future directions for research in the fields of intrusion detection and CTI, as well as how both might profit from each other. In the field of intrusion detection, already a lot of research and development have been done in the direction of signature-based IDS, with focus on black-listing. But today's evolving interconnection and the associated dependence on digital devices are the reason for sophisticated and tailored attacks, carried out by smart attackers who flexibly adjust their behavior according to the circumstances they face and therefore cannot be revealed and stopped by signature-based IDS and black-listing approaches which are only capable of revealing known attack vectors. Thus, the future research direction in cybersecurity related to big data tends toward anomaly-based intrusion detection and white-listing. The main challenges here will be reducing a usually high number of false positives that create a great effort for security analysts. Additionally, because networks grow fast and hardware devices and software versions change quickly, smart self-learning algorithms are required so that IDS can adapt themselves to new situations, without learning malicious system/network behavior and to keep the effort that has to be invested for configuration to a minimum.

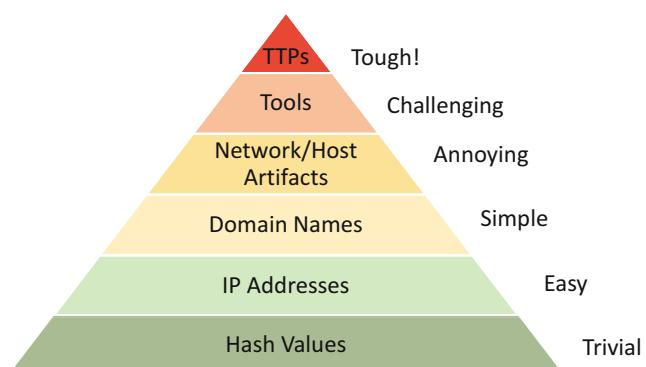
One solution to the aforementioned problems can be CTI. There exists already a considerable amount of research on how findings and observables, also from IDS outputs, can be combined

to provide insights into cybersecurity threats and attacks to finally correlate those information to gain CTI (Skopik 2017). However, since anomalies are just data without any context, and therefore do not carry any intrinsic information, the detected anomalies have to be interpreted first. To fulfill this task properly and timely, intrusion detection should be enriched with existing CTI. Thus, sharing acquired knowledge is mandatory. The “pyramid of pain” (Bianco 2014), shown in Fig. 1, demonstrates how much effort is required to acquire certain information, starting from a contextless anomaly. The goal is that stakeholders who demand a high level of security for their network infrastructures and are also able to afford the resources to analyze the large number of alerts generated by AD-based IDS solutions use their acquired knowledge to define signatures. These signatures, i.e., shareable and timely intelligence, can then be shared with further organizations on a broad basis. Hence, industry players beyond specialized security solution providers should be included in the threat intelligence creation to enable a faster reaction against new threats across whole industry sectors.

Finally, to reduce the effort of dealing with the large and quickly evolving amounts of data concerning cybersecurity, novel smart big data approaches will be required that make use of collected CTI to support and steer the configuration and deployment of cybersecurity tools, and to automatize as many tasks as possible, since the amount of generated data and the number of cyber threats and attacks keep continuously growing.

Big Data for Cybersecurity, Fig. 1

The pyramid of pain
(Bianco 2014)



Cross-References

- ▶ [Big Data in Network Anomaly Detection](#)
- ▶ [Exploring Scope of Computational Intelligence in IoT Security Paradigm](#)
- ▶ [Network Big Data Security Issues](#)

References

- Axelsson S (2000) Intrusion detection systems: a survey and taxonomy. Technical report
- Baum LE, Eagon JA (1967) An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull Am Math Soc* 73(3):360–363
- Bianco D (2014) The pyramid of pain. detectrespond. Blogspot. com/2013/03/the-pyramid-of-pain. html
- Buczak AL, Guven E (2016) A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun Surv Tutorials* 18(2): 1153–1176
- Cannady J (1998) Artificial neural networks for misuse detection. In: National information systems security conference, pp 368–381
- Chairman of the Joint Chiefs of Staff (2013) Joint publication 2-0. Joint intelligence. Technical report. http://www.dtic.mil/doctrine/new_pubs/jp2_0.pdf
- Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. *ACM Comput Surv (CSUR)* 41(3):15
- Chen M, Mao S, Liu Y (2014) Big data: a survey. *Mob Netw Appl* 19(2):171–209
- Dainotti A, Pescapé A, Ventre G (2007) Worm traffic analysis and characterization. In: IEEE international conference on communications, ICC'07. IEEE, pp 1435–1442
- Garcia-Teodoro P, Diaz-Verdejo J, Maciá-Fernández G, Vázquez E (2009) Anomaly-based network intrusion detection: techniques, systems and challenges. *Comput Secur* 28(1):18–28
- Goldstein M, Uchida S (2016) A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS One* 11(4):e0152173
- Heckerman D et al (1998) A tutorial on learning with bayesian networks. *Nato Asi Ser D Behav Soc Sci* 89:301–354
- James P Anderson (1980) Computer security threat monitoring and surveillance. Technical report 17. James P. Anderson Company, Fort Washington, DC
- Kim J, Radhakrishnan S, Dhall SK (2004) Measurement and analysis of worm propagation on Internet network topology. In: Proceedings of 13th international conference on computer communications and networks, ICCCN 2004. IEEE, pp 495–500
- Liao HJ, Lin CHR, Lin YC, Tung KY (2013) Intrusion detection system: a comprehensive review. *J Netw Comput Appl* 36(1):16–24
- NIST (2016) Guide to cyber threat information sharing. Special publication 800–150. Technical report
- OASIS (2017) Structured threat information expression v2.0. <https://oasis-open.github.io/cti-documentation/>
- Sabahi F, Movaghari A (2008) Intrusion detection: a survey. In: 3rd international conference on systems and networks communications, ICSNC'08. IEEE, pp 23–26
- Safavian SR, Landgrebe D (1991) A survey of decision tree classifier methodology. *IEEE Trans Syst Man Cybern* 21(3):660–674
- Scarfone K, Mell P (2007) Guide to intrusion detection and prevention systems (IDPS), NIST special publication, vol 800. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, p 94
- Scarfone K, Mell P (2009) An analysis of CVSS version 2 vulnerability scoring. In: Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement. IEEE Computer Society, pp 516–525
- Skopik F (2017) Collaborative cyber threat intelligence: detecting and responding to advanced cyber attacks at the national level. CRC Press, Boca Raton
- Steinwart I, Christmann A (2008) Support vector machines. Springer, New York
- Tankard C (2011) Advanced persistent threats and how to monitor and deter them. *Netw Secur* 2011(8):16–19
- Vacca JR (2013) Managing information security. Elsevier, Amsterdam/Boston/Heidelberg
- Whitman ME, Mattord HJ (2012) Principles of information security, Course technology, 4th edn. Cengage Learning, Stamford, Conn., oCLC: 930764051
- Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Cambridge
- Xu R, Wunsch D (2005) Survey of clustering algorithms. *IEEE Trans Neural Netw* 16(3):645–678

Big Data for Health

Valerio Persico

University of Napoli “Federico II”, Naples, Italy

Synonyms

[Healthcare](#); [Well-being](#)

Definitions

Health is a state in which a person is not suffering from any illness and is feeling well. *Healthcare* refers to the various services for the prevention or treatment of illness and injuries on a comprehensive, ongoing basis. Leveraging the potential of ICTs and big data allows for addressing raising issues in the health sector, providing novel technological solutions for healthcare.

Overview

Healthcare is among the most important social and economic challenges that every country faces. Indeed, constant population growth influences healthcare demand, requiring more advanced scientific solutions to address raising issues. In fact, the growing cost of medical care impacts quality of life, and this issue is exacerbated in the case of chronic diseases. Today, clinicians, administrators, as well as researchers experience more and more pressure due to the growing expectations from both the private and the public sector. Leveraging big data paves the way to unprecedented opportunities for both reducing costs and improving quality of healthcare services (Murdoch and Detsky 2013).

Understanding the progress of information and communication technologies (ICTs) is crucial to be properly aware of the potential of big data in the health sector, as ICTs have allowed to improve access, efficiency, quality, and effectiveness of healthcare-related processes. At the beginning of the 1990s, the *e-health* concept was coined, broadly defining the application of ITCs – above all, the Internet – to the health sector. It has come in common use and has been subjected to both great public interest and unprecedented levels of research effort and investments, thus allowing e-health services to benefit from the uninterrupted development of novel technological solutions. As a result, over the years the specific characterization of the e-health concept has progressively evolved and specialized, according to the widespread range of opportunities raised by the evolution of the ICTs.

The advancements in broadband communications, the unprecedented spread of wireless and mobile communication technologies, as well as the raise of *mobile apps* allowed to implement the *anywhere-and-anytime* connectivity vision, which has become a solid reality for a growing number of scenarios. These technology advancements, together with mass market demand, led to the spread of *low-cost sensing devices*, suitable for a number of different goals. Such availability of computing resources at lower costs and higher integration scale fueled the *Internet of Things* paradigm, implementing the interconnection of uniquely identifiable smart objects and devices with advanced connectivity. The circumstances above also allowed huge-scale datacenters to be deployed all over the globe and accessible to the general public through the *cloud computing paradigm*.

The success of each of the technologies and paradigms mentioned above is utmost for supporting novel healthcare applications and services. Indeed, their fruitful mixture is reshaping modern healthcare, not only with technological prospects but also with promising economic and social ones: opportunities for gathering health-related big data from a growing number of sources are provided, together with continuous advancements in technological solutions for manipulating and capitalizing it.

Coherently, the scientific literature also highlights non-negligible changes in how the concept of e-health itself is intended. Indeed, riding the wave of the technological progress, a number of healthcare-related paradigms have been introduced emphasizing the beneficial impact of big data. *Mobile health*, *smart health*, and *personalized health* are compelling examples (Silva et al. 2015; Holzinger et al. 2015; Suzuki et al. 2013; Cowie et al. 2015; Viceconti et al. 2015; Omanović-Mikličanin et al. 2015). These paradigms are commonly referred to in the scientific literature and witness how big data play a primary role in the health sector:

- Mobile health, generally aiming at delivering healthcare services regardless of any mobility

- constraints, has found fundamental pillars in smartphones and mobile apps, these tools being able to gather and deliver different kinds of information (e.g., streams of physiological signs monitoring results or public health information);
- Smart health integrates ideas from ubiquitous computing and ambient intelligence, where related processes generate large amounts of data being collected over a network under daily life and that is potentially valuable to monitor, predict, and improve patients' physical and mental conditions;
 - Personalized health, sometimes referred to as *adaptive health*, is *user-centric*, i.e., it targets to take patient-specific decisions, rather than stratifying patients into typical treatment groups.

The remainder of the entry is composed of three main parts discussing, respectively:

- the main sources and types of information that compose the health-related big data scenario;
- the most significant use cases and applications of big data in the health domain;
- the related opportunities and raising issues.

Characterization of Health-Related Big Data

Larger amounts of data are becoming available to organizations (working at global, national, regional level) both gathered from novel applications (e.g., health and wellness monitoring ones) and obtained by applying analysis techniques on already existing – but not fully capitalized yet – information (e.g., clinical reports). Worldwide digital healthcare data was estimated to be equal to 500 petabytes in 2012 and is expected to reach 25,000 petabytes in 2020 (Cyganek et al. 2016; Sun and Reddy 2013). Being aware of the sources as well as types and characteristics of this data is fundamental for understanding the applications that rely on it as well as the unclosed opportunities.

Healthcare-related big data can come from both *sources internal to the organization* (such as health records and clinical decision support systems) and *sources external to the organizations* (such as government sources, laboratories, pharmacies, insurance companies, and health maintenance organizations).

Two primary types of health-related big data can be identified: *structured data*, which refer to those with defined data model (e.g., clinical databases), and *unstructured data*, with no defined schema or explicit semantics (e.g., natural language text as in doctor prescriptions). The latter require specialized analytical methods to extract the contained information and to transform it into computable form. However, it is worth noting that this categorization has not to be intended as absolute: for instance, structured databases may contain unstructured information in practice (e.g., free-text fields) and unstructured data may in fact have some structure (e.g., metadata associated with an image or markup embedded in a document).

Up to 80% of healthcare data has been estimated to consist of rich unstructured data either only partially exploited or not leveraged at all. Only the remaining 20% is estimated to reside in automated systems, such as hospital, radiology, and laboratory information systems (Fernandes et al. 2012). Accordingly, technological solutions to either automatically interpret such resources or assist people tasked with interpreting these data sources are critical, as allowing to scale up the analysis and consider a much broader set of data.

The main sources of the health-related information today available can be identified in:

- clinical data;
- medical and clinical references and publications;
- *omics* data;
- streamed data;
- social networking and web data;
- business, organizational, and external data.

Each of the categories above is briefly discussed in the following.

Clinical Data

Clinical information about individual patients, such as results of the medical tests, images, descriptions from clinicians, and prescriptions, are collected through records that may assume several forms and denominations. As witnessed by the scientific literature, the most popular ones are (Garets and Davis 2006; Smolij and Dun 2006):

- *Electronic health records* (EHR);
- *Electronic medical records* (EMR);
- *Personal health records* (PHR).

EHRs – according to the ISO 20514 standard definition – are repositories of information about the health status of a subject of care. More specifically, they are *longitudinal* electronic records of patient health information, reporting episodes of care *across multiple care delivery organizations* within a community, region, or state.

EMRs are real-time patient health records with access to evidence-based decision support tools, possibly used to aid clinicians in decision-making. EMRs are used by healthcare practitioners to document, monitor, and manage healthcare delivery *within a care delivery organization*.

PHRs are layperson-comprehensible, lifelong tools for managing relevant health information, promoting health maintenance and assisting with chronic disease management. They are directly managed by the citizens (or by their legal proxy) and may be subjected to various legal limitations.

Because of the complex, heterogeneous, fast-growing nature of these possibly unstructured data representations, they should be considered as one of the most complex data objects in the information processing industry (Cyganek et al. 2016). In addition, text, images (encompassing a huge spectrum of methodologies to be acquired), audio, and video streams are also commonly produced in clinical context and therefore require the development of specific strategies for the extraction and the summarization of the information they contain (Martin-Sánchez and Verspoor 2014).

Medical and Clinical References and Publications

Although structured resources are increasingly available, the scientific literature often represents the main source of data (e.g., for studies in biomedicine), to the extent that the published biomedical literature can be considered as the primary repository of biomedical knowledge (Martin-Sánchez and Verspoor 2014).

Given the growth of the literature, processes requiring manual effort to extract and organize the information do not scale and are therefore unsustainable. Accordingly, different entities have devoted themselves to developing text mining web tools for indexing and cataloging of biomedical information, as well as helping users quickly and efficiently search and retrieve relevant publications (Lu 2011). In addition, current practices often require available data to be compared against clinical reference data developed by private organizations and public institutions.

Omics Data

Large amounts of biological data in various forms (e.g., genomics, microbiomics, proteomics, metabolomics, epigenomics, transcriptomics, etc.) are today generated and collected at an unprecedented speed and scale (Chen and Snyder 2013). Indeed, new-generation high-throughput sequencing technologies enable the processing of billions of DNA or RNA sequences per day (Mardis 2011).

The cost of acquiring and analyzing this biomedical data is dramatically decreasing with the help of technology upgrades and the development of novel hardware and software for parallel computing. Moreover, the rapid development and the integration of high-throughput technologies – which have significantly improved the way of looking at cells, tissues, and organisms in physiological and pathological conditions – and computational frameworks enable to examine biological systems at unprecedented level of detail, for instance, making large population-scale projects feasible (Schmidt and Hildebrandt 2017). Integrating the vast amount of often complementary data produced

by the different NGS applications allows to gain more significant biological insights toward a complete understanding of the mechanisms driving gene expression changes in human genetic pathologies, rather than limiting to the interpretation of single data sets (Costa et al. 2013).

Streamed Data

Personal devices – mainly in the form of smartphones – have significantly penetrated into society in a relatively short period of time. In addition, the progress in wireless sensors, mobile communication technology, and stream processing enable to design and implement body area networks (BANs) consisting of intelligent devices (e.g., low-power, miniaturized, non-invasive, and lightweight wireless sensors, attached on or implanted in the body) able to monitor either the human body functions or the surrounding environment. These devices allow to monitor people's health status in real time, generating large amounts of data (Chen et al. 2014; Ullah et al. 2012).

Thanks to the above enablers, large amounts of heterogeneous medical information have become available in healthcare organizations, since smart and connected healthcare devices are increasingly adopted and contribute to generating streams of structured and unstructured data.

Social Networking and Web Data

Social media technologies are significantly changing the practice of medicine: social media sources, such as online discussion forums, social networking sites and feeds, as well as more general web resources (e.g., blogs and even search query logs), are proving to be valuable information resources (Waxer et al. 2013). In more details, according to recent surveys, social media are reshaping the nature of health-related interactions, changing the way healthcare practitioners review medical records and share medical information.

In addition, social network perspective provides a set of methods for analyzing the structure of social entities as well as a variety of theories to explain these structures, whose study

may lead to identify local and global patterns, locate influential entities, and examine network dynamics (Wasserman and Faust 1994).

Business, Organizational, and External Data

Other data, not strictly related to health, such as administrative, billing, and scheduling information, further enrich the list of potential sources, allowing to implement studies that encompass not exclusively medical aspects.

Herland et al. in their review identified four distinct levels to categorize health data (Herland et al. 2014):

1. *molecular-level data* (e.g., used in bioinformatics);
2. *tissue-level data* (e.g., used in neuroinformatics);
3. *patient-level data* (e.g., used in clinical informatics);
4. *population data* (e.g., used in public health informatics).

According to the proposed review, each study may leverage data from one or more levels to answer questions that can be clustered according to their nature in different groups, such as human-scale biology, clinical, or epidemic scale.

In the light of the overall characterization provided in this section – more specifically, due to its volume, velocity, and variety – the data available in the healthcare field satisfies the commonly agreed requirements to be qualified as big data (Raghupathi and Raghupathi 2014). Newer forms of data (e.g., imaging, omics, and biometric sensors readings) are fueling the growth of the volume of health-related data; the *velocity* of gathered data has increased with regular monitoring processes (e.g., blood pressure readings, electrocardiograms, and glucose measurements); the *variety* of structured and unstructured data is evident, emphasizing the challenging aspects.

To be in line with a fourth characteristic introduced by some big data practitioners (i.e., *veracity*), big data, analytics, and outcomes have to be error-free. Finally, enormous *value* is envisioned in the predictive power of big data, due

to recent results in fields such as public health, science, and medicine where the combination of omics and clinical data is expected to fuel precision and personalized medicine understanding the mechanisms of diseases and accelerating the individualization of medical treatments (Costa 2014)

Main Applications of Big Data in the Health Sector

According to [Murdoch and Detsky](#), big data may advance the mission of healthcare delivery in four different ways (Murdoch and Detsky 2013):

- expanding the capacity to generate new knowledge;
- helping with knowledge dissemination;
- translating personalized medicine initiatives into clinical practice;
- allowing for a transformation of health care by delivering information directly to patients.

In more general terms, advances in big data analytics help naturally transform research questions from being *descriptive* (e.g., *what has happened?*) to *predictive* (e.g., *what could happen?*) and *prescriptive* (e.g., *what should we do then?*) (Chang and Choi 2016). For instance, big data analytics in healthcare can contribute to evidence-based medicine, genomic analysis, pre-adjudication fraud analysis, device remote monitoring, and patient profile analyses.

As research advances, the scientific literature witnesses how big data science is able to improve quality, efficacy, as well as efficiency of healthcare services. In the following, a number of examples are reported, organized in the categories reported below:

- genomics;
- clinical decision support;
- personalized medicine;
- health and wellness monitoring;
- medical image processing.

Genomics

Analytics related to omics sequencing techniques is an inherently big data problem: for instance, sequencing procedures produces millions and billions of reads, omics data are highly diverse, while human genome consists of 21,000 protein-coding genes (Lander 2011).

The availability of these advanced tools has enabled researchers to study genetic markers over a wide range of population and to investigate genetic causes of physiological and pathological phenotypes, investigating cell/tissue/organism identities and molecular mechanisms of human diseases, also reducing the timescale of related analyses and improving reliability. Indeed, high-throughput sequencing technologies allow to improve process efficiency by more than five orders of magnitude since sequencing of the human genome was completed.

Big data applications in genomics cover a wide variety of topics, although this field is still in a nascent stage with applications in specific focus areas (e.g., cancer), because of cost and time (Belle et al. 2015).

For instance, delivering recommendations in a clinical setting also requires fast analysis of genome-scale big data in a reliable manner, and information derived through big data analytics applied to genetic data helps in delivering personalized care to patients.

Clinical Decision Support

Clinical decision support systems (CDSSs) are designed to impact clinician decision-making about individual patients and in real time. These systems have gained popularity with the increased focus on the prevention of medical errors and have been proposed as a key element to improve patient safety. The adoption of mobile devices and apps provides additional benefits in supporting decision-making, increasing access to point-of-care tools, and improving patient outcomes (Ventola 2014).

[Berner and La Lande](#) provided an overview of CDSSs, summarizing data about the use and the impact of this kind of systems in practice (Berner and La Lande 2007). They highlight how CDSSs

have the potential to change the way medicine is taught and practiced.

In this context, the adoption of data mining techniques facilitates unprecedented understanding of large healthcare data sets, helping researchers gain both novel and deep insights. Thus, new biomedical and healthcare knowledge can be uncovered, for instance, generating scientific hypotheses from large experimental data, clinical database, as well as scientific literature to support clinical decisions but also those related to administrative settings (Yoo et al. 2012).

In addition, the use of modeling techniques (e.g., logistic regression models, Bayesian networks, support vector machines, neural networks, and classification trees, etc.) has become highly developed. The obtained models are often leveraged to support the computation of risk scores (Martinez-Millana et al. 2015). However, the performance of models is subjected to various statistical and clinical factors (e.g., deficiencies in the baseline data or modeling methods used in the study, over-fitting, differences between patient characteristics, measurement methods, healthcare systems particularities, or data quality).

Personalized Medicine

The availability of valuable big data has a fundamental importance in the implementation of the personalized health paradigm that allows to put into practice the idea of personalized care and personalized medicine.

In more particulars, *P4 medicine* is radically changing the idea of medicine itself, as it is intended to be *predictive, preventive, personalized, and participatory* (Sobradillo et al. 2011; Hood and Flores 2012)

It is based on the comprehensive understanding that the biology of each individual – in contrast to the current approach of clustering patients into treatments groups according to their phenotype – aims at tailoring treatments to its characteristics, considering the genetic information of each individual as the main data source.

A comprehensive understanding of an individual's own biology (e.g., *personalized omics*) is

expected to address both health and disease and to impact on predisposition, screening, diagnosis, prognosis, pharmacogenomics, and surveillance (Chen and Snyder 2013).

This approach is expected to cut global health budgets, both by reducing the need for hospitalization and related costs and by minimizing the unnecessary adoption of drugs. This also leads to reducing the related side effects and avoiding the development of resistance to drugs (Nice 2016).

In this scenario, leveraging proper artificial intelligence solutions to handle and analyze the huge amount of available health data is crucial (Raghupathi and Raghupathi 2014).

Health and Wellness Monitoring

In the recent past, as physiological signal monitoring devices have become ubiquitous (e.g., IoT devices), there has been an increase in utilizing telemetry and continuous physiological time series monitoring to improve patient care and management (Belle et al. 2015). These solutions are carrying unprecedented opportunities for personalized health and wellness monitoring and management, allowing for the creation and the delivery of new types of cloud services (Jovanov and Milenkovic 2011) that can either replace or complement existing hospital information systems (Kulkarni and Ozturk 2011).

Streaming data analytics in healthcare consists in systematically using continuous time-varying signals, together with related medical record information. This information is usually analyzed through statistical, quantitative, contextual, cognitive, and predictive analytical disciplines. Enriching the data consumed by analytics makes the system more robust helping balance the sensitivity and specificity of the predictive analytics.

This kind of applications requires a platform with enough bandwidth to handle multiple data sources to be designed for streaming data acquisition and ingestion. According to (Patel et al. 2012), systems for patients' remote monitoring consist of three main building blocks:

- the sensing and data collection hardware to gather physiological and movement data (e.g., tiny either on-body or in-body biosensors and battery-free RFID tags);
- the communication hardware and software to relay data to a remote center;
- the data analysis techniques to extract clinically relevant information from physiological and movement data.

Several monitoring solutions gather clinical information (such as position, temperature, or breath frequency) via body sensors or mobile devices and integrate it in the cloud, leveraging cloud characteristics such as scalable processing capability, seemingly infinite storage, and high availability (Santos et al. 2016; Thilakanathan et al. 2014; Hossain and Muhammad 2015). Often, the cloud represents a flexible and affordable solution to overcome the constraints generated by either sensor technologies or mobile devices, and therefore a growing number of proposals take advantage of the cloud capabilities to remotely off-load computation- or data-intensive tasks or to perform further processing activities and analyses (Zhang et al. 2014; Tong et al. 2014; Chen 2014; Wang et al. 2014; Wan et al. 2013).

A variety of signal processing mechanisms can be utilized to extract a multitude of target features. Their specifics largely depend on the type of diseases under investigation. Integrating these dynamic streamed data with static data from the EHR is a key component to provide situational and contextual awareness for the analytics engine.

Medical Image Processing

Medical images are a frequent and important source of data used for diagnosis, therapy assessment, and planning. They provide important information on anatomy and organ function in addition to detecting diseases states. From a data dimension point of view, medical images might have 2, 3, or 4 dimensions, as medical imaging encompasses a wide spectrum of methodologies to acquire images (e.g., computed tomography, magnetic resonance

imaging, X-ray, molecular imaging, ultrasound, photoacoustic imaging, fluoroscopy, positron emission tomography-computed tomography, and mammography). The volume of medical images is growing exponentially, and therefore such data requires large storage capacities.

As both the dimensionality and the size of data increase, understanding the dependencies among the data and designing efficient, accurate, and computationally effective methods demand new computer-aided techniques and platforms. Often, image processing techniques such as enhancement, segmentation, and denoising in addition to machine learning methods are needed. In addition, efforts have been made for collecting, compressing, sharing, and anonymizing medical data.

The integration of medical images with other types of electronic health record (e.g., EHRs) data and genomic data can also improve the accuracy and reduce the time taken for a diagnosis.

Opportunities and Issues

An increasing number and variety of organizations are beginning to apply big data to address multiple healthcare challenges, transforming data to information, supporting research, and assisting providers to improve patient care. Notwithstanding the unprecedented benefits and opportunities, the adoption of big data analytics in the health scenario is strictly related to (and even amplifies) a number of typical ICT issues, extending them to this critical domain. Therefore, healthcare practitioners are commonly facing difficulties related to managing and capitalizing this huge amount of heterogeneous data to their advantage. According to the scientific literature, data privacy and security, system design and performance, data and systems heterogeneity, and data management are the most recurring issues to be addressed.

Considering the only recent emergence of big data analytics in healthcare, governance issues including security, privacy, and ownership have yet to be fully addressed (Jee and Kim 2013; Kruse

et al. 2016). Security, privacy, and confidentiality of the data from individuals are major concerns.

The availability of real-time analytics procedures is a key requirement for a number of applications, and it is often achieved relying on cloud platforms, able to provide proper reliability with uninterrupted services and minimum down-times (Ahuja et al. 2012).

Proper data management represents an open issue. After data generation, costs are incurred when for storing, securing, transferring, as well as analyzing it (Kruse et al. 2016). Indeed, data generation is inexpensive compared with the storage and transfer of the same (Costa 2014). The lack of proper skills further exacerbates these issues representing a significant barrier to the implementation of big data analytics in health domain.

While from the one hand the adoption of cloud-based solutions mitigates performance-related issues (also providing a good trade-off with costs), from the other hand, it raises concerns about data privacy and communication confidentiality because of the adoption of external third-party's softwares and infrastructures and related concerns of losing control over data (Ermakova et al. 2013).

Data heterogeneity and lack of structure are commonly considered obstacles and still represent open issues, as healthcare data is rarely standardized, often fragmented, or generated in legacy IT systems with incompatible formats. EHRs are known to not share well across organizational lines (Kruse et al. 2016). In more particulars, biological and medical data are extremely heterogeneous (more than information from any other research field) (Costa 2014). Additional challenges arise when filling the gap emerging between the low-level sensor output and the high-level user requirements of the domain experts. As a result, continuous data acquisition and data cleansing require to be performed.

Finally, health-related data analytics is also accompanied by significant ethical challenges, ranging from risks to individual rights (e.g., concerns about autonomy) to demand of transparency and trust (e.g., in public health practice) (Vayena et al. 2015).

The adoption of big data in the health field is very promising and is gaining more and more popularity, as it is able to provide novel insights from very large data sets as well as improving outcomes while reducing costs. Its potential is great but a number of challenges remain to be addressed.

Cross-References

- ▶ [Big Data Analysis in Bioinformatics](#)
- ▶ [Big Data in the Cloud](#)
- ▶ [Big Data Technologies for DNA Sequencing](#)
- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Data Fusion](#)
- ▶ [Privacy-Preserving Data Analytics](#)
- ▶ [Secure Big Data Computing in Cloud: An Overview](#)

References

- Ahuja SP, Mani S, Zambrano J (2012) A survey of the state of cloud computing in healthcare. *Netw Commun Technol* 1(2):12
- Belle A, Thiagarajan R, Soroushmehr S, Navidi F, Beard DA, Najarian K (2015) Big data analytics in healthcare. *BioMed Res Int* 2015:370194
- Berner ES, La Lande TJ (2007) Overview of clinical decision support systems. Springer, New York, pp 3–22. https://doi.org/10.1007/978-0-387-38319-4_1
- Chang H, Choi M (2016) Big data and healthcare: building an augmented world. *Healthcare Inf Res* 22(3):153–155
- Chen M (2014) Ndnc-ban: supporting rich media healthcare services via named data networking in cloud-assisted wireless body area networks. *Inf Sci* 284:142–156
- Chen M, Mao S, Liu Y (2014) Big data: a survey. *Mobile Netw Appl* 19(2):171–209
- Chen R, Snyder M (2013) Promise of personalized Omics to precision medicine. *Wiley Interdiscip Rev Syst Biol Med* 5(1):73–82
- Costa FF (2014) Big data in biomedicine. *Drug Discov Today* 19(4):433–440
- Costa V, Aprile M, Esposito R, Ciccodicola A (2013) RNA-Seq and human complex diseases: recent accomplishments and future perspectives. *Eur J Hum Genet* 21(2):134–142

- Cowie MR, Bax J, Bruining N, Cleland JGF, Koehler F, Malik M, Pinto F, van der Velde E, Vardas P (2015) e-health: a position statement of the European society of cardiology. *Eur Heart J* 37(1):63–66. <https://doi.org/10.1093/eurheartj/ehv416>, <http://eurheartj.oxfordjournals.org/content/37/1/63>, <http://eurheartj.oxfordjournals.org/content/37/1/63.full.pdf>
- Cyganek B, Graña M, Krawczyk B, Kasprzak A, Porwik P, Walkowiak K, Woźniak M (2016) A survey of big data issues in electronic health record analysis. *Appl Artif Intell* 30(6):497–520
- Ermakova T, Huenges J, Erek K, Zarnekow R (2013) Cloud computing in healthcare – a literature review on current state of research. In: Proceedings of AMCIS 2013
- Fernandes LM, O'Connor M, Weaver V (2012) Big data, bigger outcomes. *J AHIMA* 83(10):38–43
- Garets D, Davis M (2006) Electronic medical records vs. electronic health records: yes, there is a difference. Policy white paper Chicago, HIMSS Analytics, pp 1–14
- Herland M, Khoshgoftaar TM, Wald R (2014) A review of data mining using big data in health informatics. *J Big Data* 1(1):2. <https://doi.org/10.1186/2196-1115-1-2>
- Holzinger A, Röcker C, Ziefle M (2015) From smart health to smart hospitals. Springer, Cham, pp 1–20. https://doi.org/10.1007/978-3-319-16226-3_1
- Hood L, Flores M (2012) A personal view on systems medicine and the emergence of proactive {P4} medicine: predictive, preventive, personalized and participatory. *New Biotech* 29(6): 613–624. <https://doi.org/10.1016/j.nbt.2012.03.004>, <http://www.sciencedirect.com/science/article/pii/S1871678412000477>
- Hossain MS, Muhammad G (2015) Cloud-assisted speech and face recognition framework for health monitoring. *Mobile Netw Appl* 20(3):391–399
- Jee K, Kim GH (2013) Potentiality of big data in the medical sector: focus on how to reshape the healthcare system. *Healthcare Inf Res* 19(2):79–85
- Jovanov E, Milenkovic A (2011) Body area networks for ubiquitous healthcare applications: opportunities and challenges. *J Med Syst* 35(5):1245–1254
- Kruse CS, Goswamy R, Raval Y, Marawi S (2016) Challenges and opportunities of big data in health care: a systematic review. *JMIR Med Inf* 4(4):e38
- Kulkarni P, Ozturk Y (2011) mPHASiS: mobile patient healthcare and sensor information system. *J Netw Comput Appl* 34(1):402–417. <https://doi.org/10.1016/j.jnca.2010.03.030>, <http://www.sciencedirect.com/science/article/pii/S1084804510000652>
- Lander ES (2011) Initial impact of the sequencing of the human genome. *Nature* 470(7333):187–197
- Lu Z (2011) Pubmed and beyond: a survey of web tools for searching biomedical literature. *Database* 2011:baq036. <https://doi.org/10.1093/database/baq036>, http://oup.backfile/content_public/journal_database/2011/10.1093/database/baq036/2/baq036.pdf
- Mardis ER (2011) A decade's perspective on DNA sequencing technology. *Nature* 470(7333):198–203
- Martin-Sánchez F, Verspoor K (2014) Big data in medicine is driving big changes. *Yearb Med Inform* 9(1):14–20. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4287083/>, pMID: 25123716
- Martinez-Millana A, Fernández-Llatas C, Sacchi L, Segagni D, Guillén S, Bellazzi R, Traver V (2015) From data to the decision: a software architecture to integrate predictive modelling in clinical settings. In: 2015 37th annual international conference of the IEEE engineering in medicine and biology society (EMBC), IEEE, pp 8161–8164
- Murdoch T, Detsky A (2013) The inevitable application of big data to health care. *JAMA* 309(13):1351–1352. <https://doi.org/10.1001/jama.2013.393>, http://data/journals/jama/926712/jyp130007_1351_1352.pdf
- Nice EC (2016) From proteomics to personalized medicine: the road ahead. *Expert Rev Proteomics* 13(4):341–343. <https://doi.org/10.1586/14789450.2016.1158107>, pMID: 26905403
- Omanović-Mikličanin E, Maksimović M, Vujović V (2015) The future of healthcare: nanomedicine and internet of nano things. *Folia Medica Facultatis Medicinae Universitatis Sarajevisiensis* 50(1): 23–28
- Patel S, Park H, Bonato P, Chan L, Rodgers M (2012) A review of wearable sensors and systems with application in rehabilitation. *J Neuroeng Rehabil* 9(1):1
- Raghupathi W, Raghupathi V (2014) Big data analytics in healthcare: promise and potential. *Health Inf Sci Syst* 2(1):1
- Santos J, Rodrigues JJ, Silva BM, Casal J, Saleem K, Denisov V (2016) An iot-based mobile gateway for intelligent personal assistants on mobile health environments. *J Netw Comput Appl* 71:194–204. <https://doi.org/10.1016/j.jnca.2016.03.014>, <http://www.sciencedirect.com/science/article/pii/S1084804516300273>
- Schmidt B, Hildebrandt A (2017) Next-generation sequencing: big data meets high performance computing. *Drug Discov Today* 22(4):712–717
- Silva BM, Rodrigues JJ, de la Torre Díez I, López-Coronado M, Saleem K (2015) Mobile-health: a review of current state in 2015. *J Biomed Inf* 56:265–272. <https://doi.org/10.1016/j.jbi.2015.06.003>, <http://www.sciencedirect.com/science/article/pii/S1532046415001136>
- Smolij K, Dun K (2006) Patient health information management: searching for the right model. *Perspect Health Inf Manag* 3(10):1–11
- Sobradillo P, Pozo F, Agustí Á (2011) P4 medicine: the future around the corner. *Archivos de Bronconeumología (English Edition)* 47(1):35–40. [https://doi.org/10.1016/S1579-2129\(11\)70006-4](https://doi.org/10.1016/S1579-2129(11)70006-4), <http://www.sciencedirect.com/science/article/pii/S1579212911700064>
- Sun J, Reddy CK (2013) Big data analytics for healthcare. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'13, ACM, New York, pp 1525–1525. <http://doi.acm.org/10.1145/2487575.2506178>

- Suzuki T, Tanaka H, Minami S, Yamada H, Miyata T (2013) Wearable wireless vital monitoring technology for smart health care. In: 2013 7th international symposium on medical information and communication technology (ISMICT), pp 1–4. <https://doi.org/10.1109/ISMICT.2013.6521687>
- Thilakanathan D, Chen S, Nepal S, Calvo R, Alem L (2014) A platform for secure monitoring and sharing of generic health data in the cloud. *Futur Gener Comput Syst* 35:102–113
- Tong Y, Sun J, Chow SS, Li P (2014) Cloud-assisted mobile-access of health data with privacy and auditability. *IEEE J Biomed Health Inf* 18(2):419–429
- Ullah S, Higgins H, Braem B, Latre B, Blondia C, Mograman I, Saleem S, Rahman Z, Kwak KS (2012) A comprehensive survey of wireless body area networks. *J Med Syst* 36(3):1065–1094. <https://doi.org/10.1007/s10916-010-9571-3>
- Vayena E, Salathé M, Madoff LC, Brownstein JS (2015) Ethical challenges of big data in public health. *PLoS Comput Biol* 11(2):e1003,904
- Ventola CL (2014) Mobile devices and apps for health care professionals: uses and benefits. *PT* 39(5):356–364
- Viceconti M, Hunter P, Hose R (2015) Big data, big knowledge: big data for personalized healthcare. *IEEE J Biomed Health Inf* 19(4):1209–1215
- Wan J, Zou C, Ullah S, Lai CF, Zhou M, Wang X (2013) Cloud-enabled wireless body area networks for pervasive healthcare. *IEEE Netw* 27(5):56–61
- Wang X, Gui Q, Liu B, Jin Z, Chen Y (2014) Enabling smart personalized healthcare: a hybrid mobile-cloud approach for ecg telemonitoring. *IEEE J Biomed Health Inf* 18(3):739–745
- Wasserman S, Faust K (1994) Social network analysis: methods and applications, vol 8. Cambridge University Press, Cambridge
- Waxer N, Ninan D, Ma A, Dominguez N (2013) How cloud computing and social media are changing the face of health care. *Phys Exec* 39(2):58–60
- Yoo I, Alafaireet P, Marinov M, Pena-Hernandez K, Gopidi R, Chang JF, Hua L (2012) Data mining in healthcare and biomedicine: a survey of the literature. *J Med Syst* 36(4):2431–2448. <https://doi.org/10.1007/s10916-011-9710-5>
- Zhang K, Liang X, Baura M, Lu R, Shen XS (2014) PHDA: a priority based health data aggregation with privacy preservation for cloud assisted WBANs. *Inf Sci* 284:130–141

Big Data Hardware Acceleration

► Computer Architecture for Big Data

Big Data in Automotive Industry

Awais Akram Mughal

Beijing Institute of Technology, Beijing, China

Definitions

Big Data in the automotive industry is an effort to highlight the significance and relation of Big Data Analytics in the emerging automotive industry. Automotive industry is no more a property of conventional original equipment manufacturers (OEMs), but IT companies are pouring into the industry and giving tough challenge to big brands. Big Data has played a significant role in a wide range of areas, for example, training algorithms for autonomous driving, data clustering, consumer studies, understanding the relationship between customer demand and technology impact, evaluation for better future strategies for R&D and sales, etc.

Overview

The arrival of the fourth industrial revolution has brought significant challenges for the industry (Bloem et al. 2014). Numerous industries which thrived in the past now face mere extinction. One such example is the emergence of touch screens in the mobile phones. This innovation literally wiped out numerous cellular OEMs when they failed to predict the change and couldn't meet the customer expectations in screen size, features, and quality. Automotive OEMs are no exception and are also exposed to similar challenges at all times. They are racing against fast-changing customer expectations from greater horsepower to more greener and cutting-edge disruptive technologies (Bloem et al. 2014; Deloitte 2014).

It is often overwhelming for the businesses to tackle large amounts of data to better identify, evaluate, and meet customer expectations. Big Data has proved to be a vital tool in handling such information and providing meaningful data

clusters that help lead to better strategies and products. Automotive OEMs often analyze vast amounts of data to understand their customer priorities when purchasing a car. In the recent few years, some of the other key developing technologies in automotive industry, namely, autonomous drive, shared mobility, connected vehicle (V2V and V2X), smart interfaces, etc., are thriving with the help of Big Data Analytics thus making it an important topic to discuss. The following sections discuss significance of Big Data for the automotive OEMs and its role in the development of emerging technologies in the automotive industry.

Automotive Industry and Big Data

The automotive industry has been experiencing a massive change and is still coping with the effects of the fourth industrialization revolution. Among others, one of the greatest challenges is the burst of information that a company has to yield to stay ahead of the market in technology, products, and consumer requirements.

Technology has brought the cars to life. Cars now communicate and interact with humans just like any other robot. Based on the connected devices and the sensory data generated, our modern cars are turning into data factories. When all this data is combined, it creates an immense value for all the players in the ecosystem.

However, when the information exceeds beyond the certain limit, then humans and machines tend to become careless and slow. That is one of the reasons why we often tend to forget traffic rules on the street. But what if the information could be fed to the driver in bits and pieces, when it is needed at the right time to enable them to make the best decisions? Still we are talking about a petabytes of data that is coming from numerous sources, and it will be even more complex to decide when and what information to process and deliver.

Some studies (Deloitte 2014) show to surprise that when the customer is making the first purchase of a car, brand image and price are the least considered factors. Among the top factors that influence a customer are usability and technology. However, for a second purchase, usually

technology becomes the most important factor. According to another study (IBM Institute of Business Value 2014), technology is going to be the most important external force that will impact the industry in the next decade.

Considering the size and the extent of the automotive industry, it would be months of hard labor to analyze and visualize such huge amount of data before coming to a conclusion (IBM Institute of Business Value 2014). However, first it is necessary to understand where this data is coming from and to understand its relationship.

Data Generation and the Need of Big Data in Automotive Industry

Although the term Big Data has attracted huge attention in the automotive world, behind this big word, there is a simple story. The above example is one simple demonstration of Big Data Analytics in the automotive industry. The Industry leaders, engineers, consulting agencies, and analysts have always been relying on the huge amount of data and information to clinch to the leading position in the market. In the past, this data however has been scattered at different locations.

Development Phase Data

In general, OEMs (and dealers) make immense efforts to collect actual field data to understand their customer better. Once a new vehicle has been developed, a single test drive can incur several terabytes of measurement data, and the amount of data during the development phase often exceeds the petabyte range (Bloem et al. 2014).

Consumer Phase Data

With the advancement in the Internet of things and the rise of shared economy, more and more devices are getting connected to cars. Traditionally these devices would be biometric wearables, home appliances, and audio-visual equipment. Introducing connectivity to automotive has opened new horizons for the services in the car like GPS navigation, e-mail, and live music streaming to mention a few.

The services sector around the car has evolved into a much complex structure with many new business models

- (i) **Infotainment system:** The car of today and the future will turn into a massive infotainment industry. Online streaming, news, weather, direct mobile payments, insurances, and reservations services are few services to mention and are already a billion-dollar industry. According to some conservative estimates, an average car will produce a few terabytes of data by 2020 (McKinsey and Company 2016). This data has enabled the service providers in understanding the trend, behavior, and priorities to the extent of an individual customer.
- (ii) **Mobility service providers:** Over the past few years, e-hailing services, vehicle sharing, and automated remote parking are some of the businesses that have experienced a rapid growth with the success of shared economy. Various companies are now defining location-based services based on the huge amounts of incoming data from the car for customer safety, resource optimization, better vehicle allocation, and customer management.
- (iii) **Insurance service providers:** For a long period of time, insurance providers have been relying on the aggregated data such as the driver age, gender, brand, model, etc., to set their insurance package for car and/or driver. However *usage-based insurance* is one of the new innovations in the insurance industry. Insurance companies can now monitor the mileage and driving behavior through the onboard diagnostics (OBD) in the car. User-based insurance is one of the top five innovations related to the connected car (SAS 2015).
- (iv) **Aftersales service:** In the past the OEMs have been traditionally relying on the reactive reliability statistics to improve the reliability of the vehicle and reduce the downtime. But with the advent of the Big Data technologies, this data has become much more valuable and enabled car companies to switch to predictive analysis.

As cars are becoming more and more intelligent, failures can now be predicted through data analysis for incoming data from the car sensors and radars (Pickhard 2016).

How Big Data Is Transforming Automotive Industry

Usually accidents gets unavoidable when there are millions of over speeding drivers, faulty parts in the car, kids running into the street, or an unintentional glimpse at the phone beep. There are number of other such reasons when the driver's attention is taken away, and it may cause a hazard. In such situations, the first thing an accident-affected person would want is alerting the medical aid. The next important thing could be to inform the approaching vehicles about the incident so that they may avoid any disruption. This would have also been a critical information for the navigation system on the ambulance to take the shortest route and reach in time to save the lives at the location. At times this road information is also important for other departments, like the road and traffic management departments that design the roads and manage the traffic, to ensure smooth traffic and public safety. This data would also be of immense utility for the automaker who could have identified or predicted the problem with the part, before the incident happened.

Most of this information is of importance for the city planners. This can enable them to create a whole ecosystem by networking all these building blocks of our transportation system into a standard traffic protocol and create solutions. The overall characteristics of these solutions can be classified with three words: revenue generation, cost reduction, or safety and security enhancement. However this incoming information may be too overwhelming for the few policy-makers and thus would require uncountable man hours to sort out the logic for making a common policy that might fit for all in our transportation ecosystem.

Often these solutions are drivers for the breakthrough technology that may eventually become the trend for the future automotive industry. Some of these trends are discussed below in light of implication with the Big Data; however this development is also reliant on the other associate

technologies, called enablers, that will be discussed along the way for each case.

(a) **Development phase:** Data can be a vital resource to refine or even redefine how our future vehicles are designed. By mining data from the repair shops, data from the embedded sensors in the car, and many other sources, the engineers now understand that they can reduce the engineering cycles, optimize energy usage, and reduce development costs. This is enabling car companies to roll out new models that are reliable, sustainable, and innovative. Through data analysis, OEMs are learning the trends about the customers from various regions and preparing cars that are customized for the specific region, e.g., Chinese customers may prefer an SUV when choosing a normal passenger car. The information regarding the customer to such extent would not have been possible without the Big Data. The engineers on the other hand can now also process the data and optimize the production plan to ensure the availability of the different models within the deadlines. The OEMs can also collect the data from the car to improve the downtime of the vehicle (McKinsey and Company 2016).

(b) **Market and customer:** In the past the OEMs have been traditionally relying on the reactive reliability statistics to improve the aftersales of the car. However, these days OEMs can define an overall strategy in advance by performing the proactive diagnostics. The customer who may have decided to change a faulty part at a later time due to the wait time can now be prevented from any further traffic hazards. On the other hand, OEMs can improve life expectancy of a vehicle by providing superior parts based on the weather and terrain requirements of different regions. Based on this data, OEMs can redesign their marketing strategies according to the regions. A customer in the hot desert will have very different requirements compared to a person in the frozen mountains, and the vehicle would have very different requirements for the engine oil, fuel, or other parts. Through the customer data and feedback, it can also re-

flect the vehicle performance in different terrains and what could be the future improvements in the R&D phase. Although these benefits have been theoretically proven, but yet there is a long way to go before the infrastructure is in place.

(c) **Insurance and security:** A single car may produce huge data each day, and valuable clues regarding the performance and health of the vehicle are hidden in the data generated from the car. As for the scenario defined above, there could be major challenge for the insurance companies to correctly identify the responsibility and damage estimation. As for now the user-based insurance has proved to be a feasible solution. However, real time speed and location data generated from the car, for instance, can help determine the cause and location of the accident. Big Data analysis has significantly lowered the payments that the customers had to make to insurance companies in the past.

The data collected from the accident scene can be carefully analyzed to find reasonable innovations that could be made to ensure safety of the future user. Enhanced car safety features can save many lives that would otherwise go wasted. Through the data analysis, now companies offer strong laminated windshields that would not shatter in the case of an accident, and the modernized airbags will inflate just before the impact to prevent the driver from any head injury.

(d) **Autonomous drive:** Autonomous driving is the headline of every news article, and its value is much known. As for now the driver assistive systems, collision prevention, and lane keeping are some of the features that define our so-called autonomous vehicles. But in the future, Big Data analysis will play a vital role in making the vehicles smarter. Huge amounts of traffic and roadside assistance data was fed to the car to enable it to successfully maneuver through the densely populated cities. The arrival of fully autonomous vehicles will not only reshape the mobility but also the way we humans see the car.

The interior of the car may require to be fully redefined based on the human activities. We have already seen some of the concepts presented at the auto shows by different automakers (Al Najada and Mahgoub 2016). As the data generated in the car will no longer be just about the engine speed or location, so there may be need to redefine the data clusters (Al Najada and Mahgoub 2016).

(e) ***Connected cars and Internet of things (IoT):***

IHS Automotive forecasts that there will be 152 million actively connected cars on global roads by 2020, and it will be a 40 billion-dollar industry. This makes connected cars much more interesting for the OEMs and startups. 5G technologies, with 10 GB download speed and near-zero latency, will bring in the true connected vehicle; however 4G-edge is serving the purpose at the moment. This means that in the future, the car will be able to share the data at in real time. The incoming data can be analyzed to enable many use cases. The in-car entertainment features are already in the car for some time now. The new streaming services will be offering online games, movies, and more based on the behavior of the user to bring a customized content to the user. Syncing this information with home devices will continue to provide the customer with the same experience from the car till home. Through navigation analytics, a car will be able to actuate the connected devices at the home or in office and route the way by analyzing the most updated weather and traffic conditions, keeping in view the previous preferences of the driver. Thus, Big Data Analytics will be increasingly important part of the car once it can fully integrate IoT and realize V2X communication.

Traditional Ways to Store and Process Data in Automotive Industry

If we tend to think about the data vehicles produced, then we realize it's a very complex ecosystem. Data from cars, service providers, weather reports, and environment all need to be processed and stored simultaneously. Traditionally the automotive companies spend hundreds of man-hours

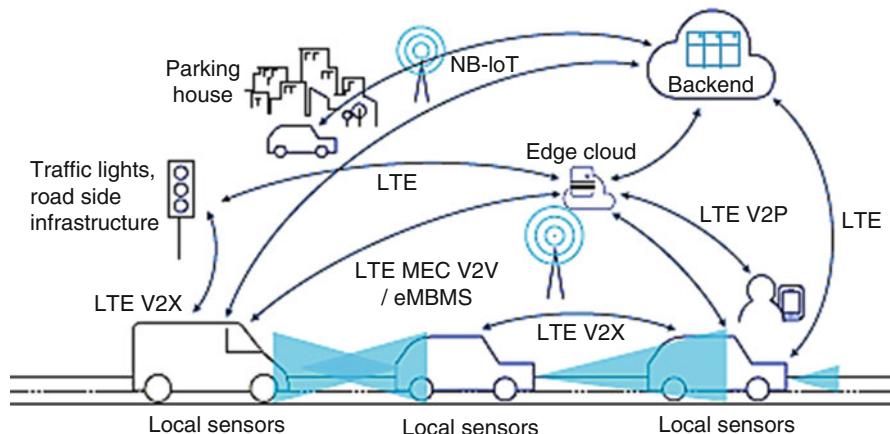
and then collect data to large servers placed at various centralized locations. These data storage techniques may work fine for the small scale ecosystems. However in future ahead with complex ecosystem of autonomous driving, requiring global monetization of car data in real time, such techniques might fail to serve. It is only then that a car manufactured in China will be able to operate fully autonomously in the USA or other parts of the world. What's needed be understood is that technology is actually removing the boundaries among different territories for the automotive.

Although the relationship of data clusters is well defined to realize the autonomous drive and for the connected cars in the future, the challenge still remains; but how do you store the data in order to retrieve and process it in the shortest viable time? Since autonomous cars require data processing in almost real time, effective Big Data processing algorithms (as in Daniel et al. (2017)) can be one of the possible solutions; however, the car industry will be looking up to even more effective means such as combination of edge and cloud computing (Liu et al. 2017) as shown in Fig. 1.

At times the sensors on board are sufficient to detect and handle certain road situations, but yet they are unable to make the car respond to the bigger ecosystem. Upcoming hazards, weather conditions, traffic jams, roadside assistance, etc., are yet to be realized through a connected network. This information is crucial and must be provided spontaneously to enable the driver to make a better decision on the road (Sherif et al. 2017). This concept proposes certain information like the humans on road and a traffic hazard directly to the edge clouds that can then provide this information to the car directly with a negligible lag. Automotive will require this and other similar concepts to not always rely on the Internet speed for better faster data processing and transfer.

Future Opportunities and Challenges for Big Data in Automotive Industry

In a complex work environment of a car, it is essential for in-vehicle measurement to be complete and time synchronization of measurement



Big Data in Automotive Industry, Fig. 1 V2X communication (Nokia 2017)

signals from different sources. The sensors and stereo cameras on the future car, for instance, will stream in large amounts of raw data during the journey. However, this incoming data must be processed in real time (Amini et al. 2017). Although, this sounds too ambitious to shorten the testing days and measurement campaigns, but it is realistic. With modern Big Data Analytics techniques, the calibration of the automotive could be done in virtual environments, and this would save hours of man time. To enhance the electronic and performance capabilities of cars, Big Data Analytics will be the most logical solution.

Another research (SAS 2015) estimated that investments in Big Data will account over \$2.8 billion in 2017 and will grow at 12% over the next 3 years. It surely has been one of the biggest enablers of the technological revolution in the automotive industry and has successfully helped to create many disruptive business models and determine new fields of innovation and research. Applying the advanced data analytics across the manufacturing value chain has generated exponential value. Well-defined sets of data have been utilized to perform analytics like the customer behavior analytics, market mix analytics, supply chain optimization analytics, and predictive quality analytics. Executives now do better process risk analytics and set new and better KPIs based on customer data accumulated globally. It enables them to make better strategies for the companies and bring stronger and more competitive products

to the market, with better technological features, through deep understanding of the global market. Big Data has enabled the globalization yet keeping the element of personalization in future cars.

Although Big Data has created several business opportunities, along come certain challenges. Technically speaking Big Data still relies on two things: proper data storage and the computational power. While using the mobility services, customers entrust mobility platforms with a considerable amount of data including their personal information and financial details, which are saved on the respective enterprise servers. For the customer it is always of utmost concern how this data is treated by these enterprises. This issue will be even more complex once the autonomous vehicles arrive and people will be riding robot taxies. Autonomous cars will be sharing numerous data content with the servers, which, if not sufficiently secured or hacked, can cause significant damage to the customer and company integrity (Sherif et al. 2017). Thus, data flowing in require proper storage and protection. The location of data storage also has direct relation to the computational power. Computation power can be evaluated from two aspects: computational speed and accuracy. Connection speed, algorithm, and data processing are among the many factors that may affect the computational speed; however, the location of data storage is also an equally important factor. Taking the data from one

location to another is often a reason of having lag between the data retrieval and processing. Can 5G completely satisfy the requirement, or should the systems be supplemented with edge cloud computing? The bigger question would be that how to prepare the automotive companies to design their automotive data management systems in a way that supplements fast and effective Big Data Analytics.

Furthermore, it is yet to be seen how Big Data evolves along with the fast-changing technological landscape. Artificial intelligence (AI) has dramatically changed the human perception about the machines and is becoming a core of future cars. Thus, it should be determined how to unite these two technologies. A bigger question would be when our autonomous car receives data through Big Data Analytics; can AI enable that car to make the right decision between colliding a tree or a child? The industry however will continue to have faith in Big Data to experiment, discover, and innovate the future opportunities.

[services/multimedia/GBE03640USEN.pdf](#). Accessed 29 Dec 2017

Liu J, Wan J, Zeng B, Wang Q, Song H, Qiu M (2017) A scalable and quick-response software defined vehicular network assisted by mobile edge computing. *IEEE Commun Mag* 55(7):94–100

McKinsey & Company (2016) Monetizing car data. <https://webcache.googleusercontent.com/search?q=cache:dzKdWumkARcJ:https://www.mckinsey.com/~media/McKinsey/Industries/Automotive%2520and%2520Assembly/Our%2520Insights/Monetizing%2520car%2520data/Monetizing-car-data.ashx+&cd=1&hl=en&ct=cInk&gl=de>. Accessed 12 Dec 2017

Nokia (2017) Nokia connected vehicles V2X. https://onestore.nokia.com/asset/200766/Nokia_Connected_Vehicles_V2X_Flyer_Document_EN.pdf. Accessed 18 Dec 2017

Pickhard F (2016) Measuring “everything” big data in automotive engineering. *ATZelektronik Worldwide* 11(1):64

SAS (2015) The connected vehicle: big data, big opportunities. https://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/connected-vehicle-107832.pdf. Accessed 25 Dec 2017

Sherif AB, Rabieh K, Mahmoud MM, Liang X (2017) Privacy-preserving ride sharing scheme for autonomous vehicles in big data era. *IEEE IoT J* 4(2):611–618

References

- Al Najada H, Mahgoub I (2016) Autonomous vehicles safe-optimal trajectory selection based on big data analysis and predefined user preferences. In: Ubiquitous computing, electronics & mobile communication conference (UEMCON), IEEE annual. IEEE, pp 1–6. <http://ieeexplore.ieee.org/abstract/document/7777922/>
- Amini S, Gerostathopoulos I, Prehofer C (2017) Big data analytics architecture for real-time traffic control. In: 2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS). IEEE, pp 710–715. <http://ieeexplore.ieee.org/abstract/document/7777922/>
- Bloem J, Doorn MV, Duivestein S, Excoffier D, Maas R, Ommeren EV (2014) The fourth industrial revolution – things to tighten the link between IT and OT. Sogeti VINT2014
- Daniel A, Subburathinam K, Paul A, Rajkumar N, Rho S (2017) Big autonomous vehicular data classifications: towards procuring intelligence in ITS. *Veh Commun* 9:306–312. ISSN: 2214-2096
- Deloitte (2014) Driving through the consumer’s mind: steps in the buying process. <https://www2.deloitte.com/content/dam/Deloitte/in/Documents/manufacturing/in-mfg-dtcm-steps-in-the-buying-process-noexp.pdf>. Accessed 12 Dec 2017
- IBM Institute of Business Value (2014) Automotive 2025: industry without borders. <https://www-935.ibm.com/>

Big Data in Cellular Networks

► Big Data in Mobile Networks

Big Data in Computer Network Monitoring

Idilio Drago¹, Marco Mellia¹, and Alessandro D’Alconzo²

¹Politecnico di Torino, Turin, Italy

²Austrian Institute of Technology, Vienna, Austria

Synonyms

Applications; Network measurements; Research directions; Survey

Overview

Network monitoring applications (e.g., anomaly detection and traffic classification) are among the first sources of *big data*. With the advent of algorithms and frameworks able to handle datasets of unprecedented scales, researchers and practitioners have the opportunity to face network monitoring problems with novel data-driven approaches. This section summarizes the state of the art on the use of big data approaches for network monitoring. It describes why network monitoring is a big data problem and how the big data approaches are assisting on network monitoring tasks. Open research directions are then highlighted.

Network Monitoring: Goals and Challenges

Monitoring and managing the Internet is more fundamental than ever, since the critical services that rely on the Internet to operate are growing day by day. Monitoring helps administrators to guarantee that the network is working as expected as well as to plan its evolution, raising alerts in case of attacks or anomalies, and assisting on the prediction of traffic loads based on historical data.

Network measurements are key for these tasks. Measurements are collected from network links, forwarding equipment and end nodes in real time by exporting either information extracted directly from the traffic (i.e., passive measurements) or from probing packets injected in the network (i.e., active measurements). A variety of methods and algorithms have been proposed to extract higher-level knowledge from the measurements, from ad hoc heuristics that maintain basic statistics describing the status of the network to advanced machine learning methodologies that detect anomalies and inform operators about problems automatically.

However, collecting and analyzing network measurements is by no means a simple task. The sheer traffic volumes, the large number of legitimate services running on the network, the widespread presence of vulnerabilities and ma-

licious traffic, and the variety of traffic sources are just some examples of the challenges to monitor the Internet. Critical monitoring applications, such as anomaly detection and traffic classification, require efficient mechanisms that allow the processing of millions of events per second – events that otherwise would be of little value if analyzed in isolation.

These challenges match well with what is nowadays used to define *big data* – i.e., data that is intrinsically large (volume) and heterogeneous (variety) that needs to be analyzed in real time (velocity) and generates knowledge crucial for those managing, operating, and planning the network (value).

Are researchers and practitioners leveraging big data approaches for network monitoring problems? What are the open challenges to fully exploit the big data potentials when monitoring the network? This section explores these questions providing a brief overview on why network monitoring can be characterized as a big data problem and how big data approaches are assisting network monitoring. Open research directions are then highlighted.

Network Monitoring: A Big Data Problem

The list of traffic monitoring applications that could be characterized as a big data problem and thus profit from big data approaches is vast. Traffic classification and anomaly detection for cybersecurity are used in the following to illustrate why network monitoring is a big data problem.

Traffic Classification

Valenti et al. (2013) define traffic classification as the problem of recognizing services generating streams of packets. Traffic classification is an essential step for many network applications, such as to perform per-service traffic engineering, traffic accounting, etc. Callado et al. (2009) and Nguyen and Armitage (2008) have categorized the several methodologies for traffic classification, grouping them according to the core classification strategy:

- (i) *Packet inspection methods* search for predefined fingerprints that characterize services (e.g., protocol messages) in packet contents. These algorithms are recognized to be generally fast and precise, but their importance has been drastically reduced by the increasing deployment of encrypted protocols;
- (ii) *Machine learning methods* automatically search for typical traffic characteristics of services. Supervised learning algorithms are usually preferred, in which features of the traffic, e.g., packet sizes and inter-arrival times, are used as input to train classification models describing the traffic. Neural networks, decision trees, and support vector machines are some examples of suitable learning algorithms. These methods are appealing since they can find complex patterns describing the services with minimal human intervention. However, they require large training datasets to build up good-quality models. Moreover, it is hard to achieve general classification models. In other words, the models that classify traffic in one network may not be able to classify traffic from other networks. Data from different networks is usually needed to build up generic models.
- (iii) *Behavioral methods* identify services based on the behavior of network nodes – e.g., servers that are typically reached by clients and the order in which servers are contacted. Also in this case, machine learning algorithms can help to discover patterns, e.g., learning rules that describe the behavior of nodes and outputting the most likely services running on the nodes. Behavioral methods require little metadata, thus working well with encrypted services. However, as before, they require large datasets for learning the hosts' behaviors, as well as training data from different networks, in particular if coupled with machine learning algorithms.

Anomaly Detection

Bhuyan et al. (2013) and García-Teodoro et al. (2009) describe diverse techniques for anomaly

detection that are often employed in cyber-security tasks. For example, anomaly detection is used in network security contexts to detect traffic related to security flaws, virus, or malware, so to trigger actions to protect users and the network.

Anomaly detection for cyber-security builds on the assumption that malicious activities change the behavior of the network traffic, which is determined by interactions with legitimate services. Anomaly detection methods rely on models that summarize the “normal” network behavior from measurements, i.e., building a baseline. Traffic classification algorithms can be leveraged for learning the behavior of legitimate services, e.g., by building behavioral models that are able to identify popular services. Live traffic is then monitored and alerts are triggered when the network behavior differs from the baseline models according to predefined metrics.

Anomaly detection methods are attractive since they allow the early detection of unknown threats, e.g., zero-day exploits. These methods however may not detect stealth attacks, which are not sufficiently large to disturb the traffic. Moreover, the large number of Internet services represents a challenge for anomaly detection, since unknown services (that are possibly legitimate) might be considered anomalies as well. In other words, anomaly detection may suffer from false positives, i.e., false alerts triggered because of unknown services or by transient changes in legitimate services.

Big Data Challenges on Network Monitoring

Network monitoring applications face all challenges of typical big data applications.

Taking *velocity* as a first example: Traffic classification and anomaly detection are often performed in high-speed networks, e.g., backbone links at Internet service providers (ISPs). The systems need to cope with tens or even hundreds of gigabits of traffic per second since thousands of users are aggregated in such links. Many scalable network monitoring approaches have been proposed to overcome this challenge. Flow monitoring is one prominent example.

Instead of letting complex algorithms handle raw traffic directly, a specialized equipment that implements basic preprocessing steps in hardware is deployed. This equipment is usually called a *flow exporter*. The flow exporter builds summaries of the traffic according to the *flows* seen active in the network. Flow exporters have been shown to scale well, processing more than 40 Gbps of network traffic with commodity servers (Trevisan et al. 2017). Analysis applications then operate on such flow-level summaries. For instance, machine learning algorithms have been shown to deliver promising results with flow-level measurements, despite the coarse granularity of the measurements – see Sperotto et al. (2010) for details.

Hofstede et al. (2014), however, show that even flow-based approaches are starting to be challenged by the ever-increasing network speeds. Table 1 illustrates the speed in which flow exporters would send out data from a vantage point when observing 50 Gbps of raw traffic on average. The table includes volumes estimated for different flow export technologies – i.e., Cisco’s NetFlow v5 and NetFlow v9 as well as IPFIX, which is the IETF standard protocol to export flow-level information from the network. The data rate leaving the flow exporter is remarkably smaller than the raw traffic stream. Nevertheless, applications processing only the flow summaries (e.g., for anomaly detection) would still have to handle more than 70 Mbps if IPFIX is chosen to export data. More than that, these numbers refer to a single vantage point, and there could be hundreds of vantage points in a midsized network. Whereas sampling is often used to reduce exported volumes (see numbers

for the 1:10 sampling rate), it reduces the *value* of the data, making it harder to run some monitoring applications.

The *volume* of measurements is another clear example on why network monitoring faces big data challenges. The availability of historical data is a strong asset for some monitoring applications, e.g., security applications that perform forensics can rely on historical data to detect whether users have been impacted by previously unknown attacks. Table 1 suggests that the archival of network measurements requires large storage capacity even if highly efficient formats such as compressed flow summaries are selected. Figure 1 quantifies this requirement by depicting the storage space needed to archive compressed flow measurements from vantage points monitoring around 30,000 end users. Note that these vantage points are relatively small, observing only few Gbps of traffic on peak periods. The figure shows that the space occupied by the measurements grows almost linearly with time as expected. After 4 years, the archival consumes almost 30 TB.

Around three billion flows have been reported by flow exporters in each month depicted in the figure. This shows how challenging is to extract *value* from network measurements. Consider an anomaly detection system that triggers an event every time a set of flows is considered suspicious. The system would likely bother network administrators with the false alerts even if only a single false alert would be generated every tens of millions flows.

Big Data in Computer Network Monitoring, Table 1

Estimated stream of packets and bytes leaving a flow exporter that observes 50 Gbps on average – source (Hofstede et al. 2014)

Sampling	Protocol	Packets/s	Bits/s
1:1	NetFlow v5	4.1 k	52 M
1:1	NetFlow v9	10.2 k	62 M
1:10		4.7 k	27 M
1:1	IPFIX	12.5 k	75 M

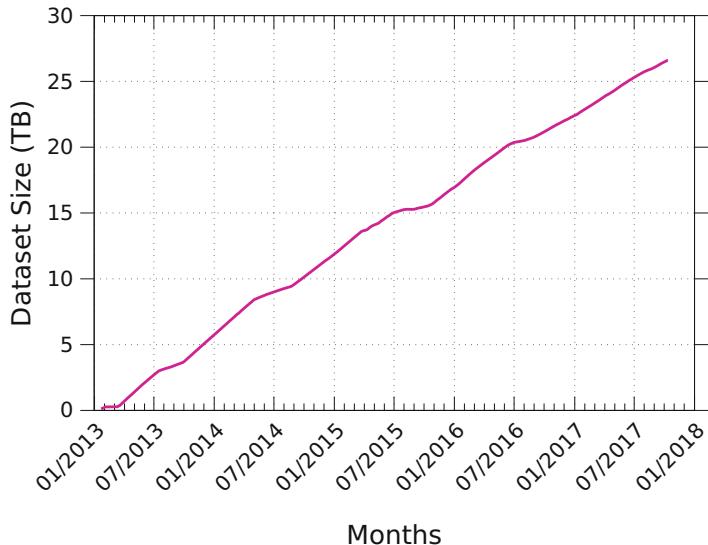
Big Data Technologies in Network Monitoring

Since network monitoring shows the characteristics of a big data problem, a natural question is whether researchers and practitioners are leveraging big data approaches in this context.

There is still a small number of academic works that actually leverage big data for network monitoring. This is somehow surprising considering that big data technologies such as

Big Data in Computer Network Monitoring,

Fig. 1 Storage to archive flow measurements from links aggregating around 30 k end users



MapReduce are rather mature – in fact, MapReduce has been introduced by Dean and Ghemawat (2004). Even considering the well-known weaknesses of some technologies, e.g., MapReduce inefficiencies when dealing with iterative algorithms over large datasets, alternatives do exist, and they are now already well-established too. A prominent example is Spark, proposed by Zaharia et al. (2010) precisely to overcome the weaknesses of MapReduce.

This raises questions on whether these technologies lack characteristics to fulfill the requirements of network monitoring applications. Roughly speaking, early adopters have mostly tried to adapt or to extend existing big data technologies to better fit the network monitoring applications.

To fully understand current efforts, it is convenient to put the ongoing research on big data approaches for network monitoring in the perspective of a knowledge discovery in data (KDD) process. The classical KDD process, attributed to Fayyad et al. (1996), is summarized as a sequence of operations over the data: gathering, selection, preprocessing, transformation, mining, evaluation, interpretation, and visualization. This schema is depicted in Fig. 2. Next works applying big data approaches for network monitoring are discussed according to the covered KDD phases, which are clustered in two groups:

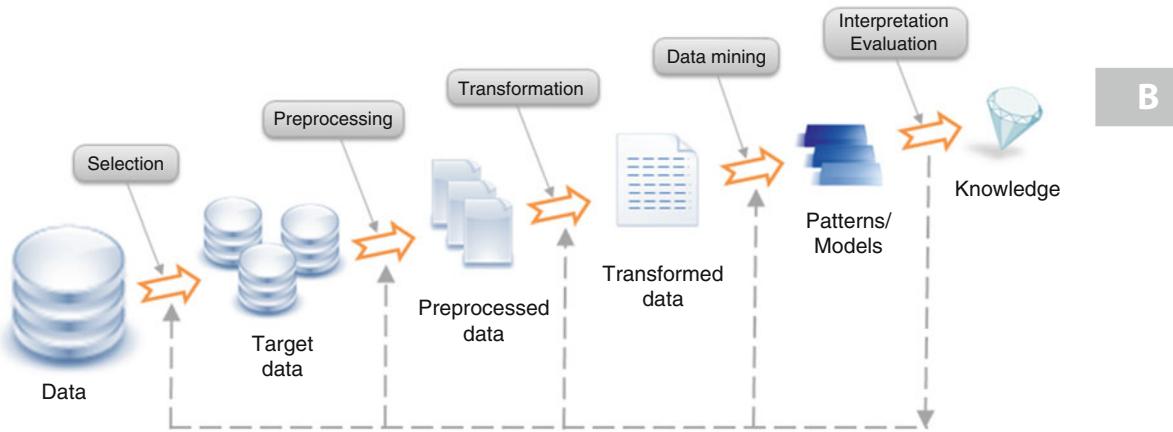
data management (from data gathering until transformation) and analytics (i.e., the remaining operations).

Data Management

Marchal et al. (2014) introduce an intrusion detection system that relies on heterogeneous sources, e.g., NetFlow, DNS logs, and honeypots. Intruders are identified by ad hoc metrics that spot anomalies (possibly malicious) in the data. Queries that are generally common in network monitoring are solved to calculate the metrics, such as for finding records containing a given string and records that pass particular filters – e.g., to count contacted destination IP addresses given a source address.

The authors benchmark different big data technologies concluding that Spark and Shark answer queries faster than alternatives (e.g., Hive). They however highlight challenges to process the measurements: There is a shortage of solutions for ingesting, reading, and consolidating measurements in the distributed file systems supported by the frameworks.

Lee and Lee (2013) have also identified these problems. Network measurements are often captured and transferred in optimized binary formats that cannot be directly and efficiently stored in distributed databases and file systems. The authors focus on how to process raw PCAP and



Big Data in Computer Network Monitoring, Fig. 2 Classical knowledge discovery in data (KDD) process – source (Fayyad et al. 1996)

NetFlow traces in Hadoop. Since HDFS does not know how to split PCAPs and NetFlow formats, the processing of each file in Hadoop-based solutions cannot profit from parallelism and data locality to scale up horizontally. Authors implement new Hadoop APIs to explicitly process PCAP and NetFlow archives, showing that the APIs improve performance.

These APIs are however specific to HDFS, PCAP, and NetFlow and allow only sequential reading of the archives in HDFS. While they certainly alleviate the problem, challenges persist if one would need advanced analytics or functionalities that are not able to interact with the APIs. For instance, when faced with similar problem of ingesting heterogeneous measurements into big data frameworks, Wullink et al. (2016) have created an extra preprocessing step to convert measurements from the original format (i.e., PCAPs in HDFS) into a query-optimized format (i.e., Parquet). This conversion is shown to bring massive improvements in performance, but only preselected fields are loaded into the query-optimized format.

These works make clear that big data frameworks are attractive to process network measurements, thanks to built-in capabilities to handle the usually large volumes of measurements. Nevertheless, they shed lights on how basic steps, like ingesting measurements into the frameworks, are by no means trivial. Those are not the only chal-

lenges. Other challenges immediately become evident when one considers that network monitoring applications often operate not only with batches of historical data (i.e., volume) but also with live measurement streams that need to be evaluated on the fly (i.e., velocity).

Bär et al. (2014) show that solutions targeting batch processing (e.g., Spark) present poor performance for such online analysis, and even stream-oriented alternatives (e.g., Spark Streaming) miss functionalities that are essential for network monitoring applications. The latter is also highlighted by Čermák et al. (2016), who mention the difficulties to recalculate metrics from old data as one of the several weaknesses in such general stream processing engines. Bär et al. (2014) propose DBStream to calculate continuous and rolling analytics from online data streams. DBStream is thus built upon some ideas of big data frameworks but specialized to solve network monitoring problems. DBStream is still a centralized system, relying on a single-node SQL database, thus lacking the horizontal scalability features of a modern big data solution.

Noting a similar gap in stream analytics tools for BGP measurements, Orsini et al. (2016) introduce BGPSstream. They point out that although BGP is a crucial Internet service that manages routing, efficient ways to process large BGP measurements on real time are missing, e.g., to detect anomalies and attacks in the routing service. BG-

PStream provides tools and libraries that connect live BGP data sources to APIs and applications. BGPSStream can be used, for instance, to efficiently ingest online data into stream processing solutions, such as on Spark Streaming.

Analytics

The management of network measurements, i.e., gathering, ingestion etc., is just the starting point of network monitoring applications. Often data analytics methods used by these applications have troubles to handle big data – e.g., some machine learning algorithms do not scale well with large datasets or are hard to parallelize.

Selecting the best algorithms for the specific network monitoring problem is a complex task. For instance, it is commonly accepted that there is no single best learning model for addressing different problems simultaneously. Indeed, even if multiple models could be very well suited for a particular problem, it is very difficult to understand which one performs best in operational network where different data distributions and statistical mixes may emerge. Recently, Vanerio and Casas (2017) have proposed a solution for network security and anomaly detection that is suitable for big data. Their method is based on the use of *super learners*, a class of ensemble learning techniques known to perform asymptotically as well as the best combination of the base learners.

Liu et al. (2014), Fontugne et al. (2014) and Wang et al. (2016) are other examples of authors that have started to implement analytics methods based on MapReduce or Spark to process large collections of measurements (e.g., feature selection). However, works on big data analytics for network monitoring are still incipient, lacking comprehensive approaches that are applicable for more sophisticated cases.

Research Directions

From this short survey, it can be concluded that researchers have identified the potential of big data approaches for network monitoring, but current technologies seem to miss features required

by network monitoring applications. It is also clear that there is work in progress to fill in this gap.

If one considers the KDD process in a big data context, there are several general big data frameworks and technologies that cover the initial stages, e.g., data gathering and preprocessing. The network monitoring community has clearly focused on these initial stages when measurements become big data. The community is on the way to answer questions such as how to ingest measurements in big data frameworks and how to process large streams of live measurements from different sources. However, these efforts are still somehow isolated. Lessons learned need to be incorporated in the frameworks, so to allow easier processing of other sources and the reuse of APIs and libraries for different network monitoring analytics. One recent attempt in that direction is Apache Spot (2017), which proposes an open data model to facilitate the integration of different measurements, providing a common taxonomy for describing security telemetry data and to detect threats.

Since both online stream and offline batch analyses are needed for network monitoring, another research direction is toward the unification of the two programming models. For example, building upon the *dataflow* abstraction (Akidau et al. 2015), Apache Beam (2017) offers one of such unified programming models. It can represent and transform both (i) batch sources and (ii) streaming sources, thus relieving programmers from the burdens of keeping two code bases and increasing portability across execution engines.

As the challenges in the initial phases of the KDD process are tackled, researchers are moving toward subsequent KDD phases, such as by proposing data mining algorithms and visualization techniques suitable for big data. The recent progresses in training deep learning models with big datasets are just some examples of this trend. However, network monitoring applications that profit from these progresses are almost nonexistent.

Casas et al. (2016) are working on Big-DAMA, one of the few initiatives that are

trying to face these challenges comprehensively. Big-DAMA aims at conceiving a new *big data analytics framework* specifically tailored to network monitoring applications. Big-DAMA will include scalable online and offline machine learning and data mining algorithms to analyze extremely fast and/or large network measurements, particularly focusing on stream analysis algorithms and online processing tasks. Big-DAMA aims also at creating new benchmarks that will allow researchers and practitioners to understand which of the many tools and techniques emerged in recent years suit best for network monitoring applications.

Projects such as Apache Spot and Big-DAMA are at their early stages and clearly there is still a long way to go. It seems however a matter of time until the network monitoring community fills in the gaps, fully exploiting the potential of big data approaches to improve network monitoring applications.

Cross-References

- ▶ [Big Data for Cybersecurity](#)
- ▶ [Big Data in Mobile Networks](#)
- ▶ [Big Data in Network Anomaly Detection](#)

References

- Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E, Whittle S (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc VLDB Endow* 8(12):1792–1803
- Apache Beam (2017) Apache Beam: an advanced unified programming model. <https://beam.apache.org/>
- Apache Spot (2017) A community approach to fighting cyber threats. <http://spot.incubator.apache.org/>
- Bär A, Finamore A, Casas P, Golab L, Mellia M (2014) Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis. In: Proceedings of the BigData, pp 165–170
- Bhuyan MH, Bhattacharyya DK, Kalita JK (2013) Network anomaly detection: methods, systems and tools. *Commun Surv Tutorials* 16(1):303–336
- Callado A, Kamienski C, Szabó G, Gero BP, Kelner J, Fernandes S, Sadok D (2009) A survey on internet traffic identification. *Commun Surv Tutorials* 11(3): 37–52
- Casas P, D’Alconzo A, Zseby T, Mellia M (2016) Big-DAMA: big data analytics for network traffic monitoring and analysis. In: Proceedings of the LANCOMM, pp 1–3
- Čermák M, Jirsík T, Laštovička M (2016) Real-time analysis of NetFlow data for generating network traffic statistics using Apache Spark. In: Proceedings of the NOMS, pp 1019–1020
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the OSDI, pp 10–10
- Fayyad UM, Piatetsky-Shapiro G, Smyth P (1996) From data mining to knowledge discovery: an overview. *AI Mag* 17(3):37–54
- Fontugne R, Mazel J, Fukuda K (2014) Hashdoop: a mapreduce framework for network anomaly detection. In: Proceedings of the INFOCOM WKSHPS, pp 494–499
- García-Teodoro P, Díaz-Verdejo J, Maciá-Fernández G, Vázquez E (2009) Anomaly-based network intrusion detection: techniques, systems and challenges. *Comput Secur* 28:18–28
- Hofstede R, Celeda P, Trammell B, Drago I, Sadre R, Sperotto A, Pras A (2014) Flow monitoring explained: from packet capture to data analysis with NetFlow and IPFIX. *Commun Surv Tutorials* 16(4):2037–2064
- Lee Y, Lee Y (2013) Toward scalable internet traffic measurement and analysis with hadoop. *SIGCOMM Comput Commun Rev* 43(1):5–13
- Liu J, Liu F, Ansari N (2014) Monitoring and analyzing big traffic data of a large-scale cellular network with hadoop. *IEEE Netw* 28(4):32–39
- Marchal S, Jiang X, State R, Engel T (2014) A big data architecture for large scale security monitoring. In: Proceedings of the BIGDATACONGRESS, pp 56–63
- Nguyen TT, Armitage G (2008) A survey of techniques for internet traffic classification using machine learning. *Commun Surv Tutorials* 10(4):56–76
- Orsini C, King A, Giordano D, Giotsas V, Dainotti A (2016) BGPStream: a software framework for live and historical BGP data analysis. In: Proceedings of the IMC, pp 429–444
- Sperotto A, Schaffrath G, Sadre R, Morariu C, Pras A, Stiller B (2010) An overview of IP flow-based intrusion detection. *Commun Surv Tutorials* 12(3): 343–356
- Trevisan M, Finamore A, Mellia M, Munafò M, Rossi D (2017) Traffic analysis with off-the-shelf hardware: challenges and lessons learned. *IEEE Commun Mag* 55(3):163–169
- Valenti S, Rossi D, Dainotti A, Pescapè A, Finamore A, Mellia M (2013) Reviewing traffic classification. In: Data traffic monitoring and analysis – from measurement, classification, and anomaly detection to quality of experience, 1st edn. Springer, Heidelberg

- Vanerio J, Casas P (2017) Ensemble-learning approaches for network security and anomaly detection. In: Proceedings of the Big-DAMA, pp 1–6
- Wang Y, Ke W, Tao X (2016) A feature selection method for large-scale network traffic classification based on spark. *Information* 7(1):6
- Wullink M, Moura GCM, Müller M, Hesselman C (2016) ENTRADA: a high-performance network traffic data stream. In: Proceedings of the NOMS, pp 913–918
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the HotCloud, pp 10–10

Big Data in Cultural Heritage

Vincenzo Moscato

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, University of Naples, Naples, Italy

Definitions

Cultural heritage information management mainly concerns the development of tools and applications for the storage, processing and retrieval of cultural digital contents, coming from distributed and heterogeneous data sources such as digital libraries and archives of cultural foundations, multimedia art collections, web encyclopedias and social media networks, and so on. In addition, if we also consider data from possible sensor networks – deployed in a given cultural environment (e.g., museums, archeological sites, old town centers, etc.) – the information to manage assumes characteristics of volume, velocity, and variety typical of big data. Thus, technologies for big data management can be properly investigated to realize a useful platform for cultural heritage applications.

Overview

In this chapter, we describe the main technological challenges and issues for the management of Big Data related to the Cultural Heritage domain.

Introduction

Nowadays, using information and communication technologies (ICTs) to enhance and promote worldwide cultural heritage (CH) surely represents a very important research issue with a variety of potential applications. In addition, we can observe how this challenge is particularly important in Italy, whose artistic heritage embodies a universal resource of priceless value, attracting every year millions of visitors to monuments, archeological sites, and museums.

Indeed, ICTs have been giving for many years and in different ways a very useful support to CH fruition and preservation (Kabassi 2013).

The initial challenge was to provide several techniques, tools, and systems to assist the annotation and cataloguing of CH objects within digital repositories that in many cases can include a large amount of contents with rich and different metadata.

Successively, other advancements of ICT have made possible to “enrich” the original information (e.g., by means of social media and augmented/virtual reality techniques), in order to improve and facilitate fruition of cultural objects, also using web and mobile applications.

As an example, the mode of fruition, and even the purpose, of CH exhibitions has radically changed in the last decade rapidly moving from an old vision, providing tourists with static information consisting of a large amount of “cultural signs,” to novel personalized services that meet the visitors’ personal tastes by considering their cultural needs and preferences (e.g., by means of the introduction of recommendation facilities Albanese et al. 2013).

More recently, the adoption of *Future Internet* technologies, particularly the paradigm of the *Internet of Things*, has allowed to realize *smart* cultural environments, whose “state” is captured by a set of sensors: from temperature/humidity sensors to social (user) devices.

We can thus imagine a novel scenario where *persons* (citizens, tourists, etc.) and *objects* (artifacts, buildings, rooms, etc.) conveniently equipped with the appropriate devices (smart-

phone, video cameras, temperature/humidity sensors, etc.) can communicate supporting a lot of cultural *context-aware* services (from multimedia storytelling of an artwork when a user is near to the related artifact, to cultural items' monitoring for preservation and security aims) and generating a very large amount of information that has to be properly managed leveraging proper infrastructures (Amato et al. 2012; Colace et al. 2015).

In particular, it is easy to note as the tremendous quantity of digital data within a smart cultural environment shows the well-known *Big Data* features, mainly due to their high change rate, their large volume, and the intrinsic heterogeneity, especially if we try to combine information coming from different cultural repositories and social media networks (Chen and Zhang 2014).

Thus, in the design of smart cultural environments, we have to necessarily contemplate big data technologies (i.e., distributed file systems, NOSQL and in-memory databases, models and engines for distributed processing, data ingestion and analytics, resource management facilities, etc.) and understand how CH information systems can leverage platforms based on such facilities to effectively and efficiently manage all the different kinds of data.

Also current work on infrastructures relevant to digital cultural objects, such as DARIAH and CLARIN, and large-scale aggregators of content, like EUROPEANA and ARIADNE, are now investigating how the use of big data technologies can improve the discussed CH fruition and preservation aspects.

In this entry, we first describe the main characteristics and functionalities required to a general big data platform for CH information management; then we present CHIS (cultural heritage information system), a big data system to query, browse, and analyze cultural digital contents from a set of distributed and heterogeneous digital repositories. In particular, the system prototype has been developed within DATABENC, the high technology district for cultural heritage management of the Campania Region, in Italy (www.databenc.it).

Functionalities and Features of a Big Data Platform for CH Applications

B

We can imagine a *smart cultural heritage environment* (e.g., museums, archeological sites, old town centers, etc.) as grounded on a set of cultural *points of interest* (PoI), which correspond to one or more cultural objects (e.g., specific ruins of an archeological site, sculptures and/or pictures exhibited within a museum, historical buildings, monuments and famous squares in an old town center, and so on) (Schaffers et al. 2011).

Consequently, a very large amount of data can be generated within such environments. In particular, we have to deal with heterogeneous information coming from different data sources:

- annotations provided by CH foundations' archives or digital libraries;
- descriptions from open encyclopedias (e.g., Wikipedia/DBpedia);
- sensor data related to environmental parameters (e.g., humidity and temperature) and captured by proper sensor networks deployed in a given PoI;
- multimedia contents – video, text, image and audio – coming from social media networks (e.g., Picasa, YouTube, and Flickr) and other kinds of digital libraries;
- social data as opinions and comments of users from common online social networks like Facebook, Twitter, etc.;
- any other kind of useful data gathered by means of web services, e.g., about other touristic attractions, or accommodations in the same geographic area, or meteorological information and so on.

In order to meet variety, velocity, and volume of all the described information related to a smart cultural environment, a general purpose CH big data platform has to be characterized by the following functionalities:

- capability to gather information from the distributed and heterogeneous data sources (e.g.,

- sensor networks, social media, digital libraries and archives, multimedia collections, etc.);
- advanced data management techniques and technologies;
 - system resources management;
 - real-time and complex event processing for monitoring and security aims;
 - ability to provide useful and personalized data to users based on their preferences/needs and context and advanced information retrieval services, data analytics, and other utilities, according to the SOA (service-oriented architecture) paradigm.

By means of a set of ad hoc APIs, and exploiting the discussed services, a CH big data platform has to be able to support several applications: mobile guides for cultural environments; web portals to promote the cultural heritage of a given organization, multimedia recommender, and storytelling systems; event recognition for cultural heritage preservation and security; and so on.

Concerning technical features, a big data platform for CH applications has to be built on a layered architecture (see Fig. 1). In the following, we describe the responsibilities of each layer.

The *resource management layer* has to coordinate and optimize the use of the different resources needed for implementing the whole system by means of the Resource Manager component.

The *data source layer* has as main goal to manage the “ingestion” of all the different data coming from the heterogeneous data sources. The data ingestion component can operate in a continuous or asynchronous way, in real-time or batched manner or both depending on the characteristics of the source. In addition, ingested data has to be then represented according to a given format.

The *stream processing layer* has to be used to provide real-time processing and event recognition activities on the data streams using a stream processing engine and a complex event processor (Cugola and Margara 2015; Panigati et al. 2015).

The *data storage and management layer* has to store and manage data of interest in a proper

knowledge base (KB) in compliance with a given data model, eventually exploiting the LD/LOD (linked data/linked open data) paradigm (possible data integration problems for heterogeneous data sources can be addressed by means of schema mapping, record linkage and data fusion techniques).

In addition, specific semantics to be attached to the data can be provided using annotation schemata, including ontologies, vocabularies, taxonomies, etc., related to the cultural heritage domain.

The KB can leverage different data repositories realized by advanced data management technologies (e.g., distributed file systems, NoSQL, in-memory and relational databases) and provides a set of basic APIs to read/write data by an access method manager.

The *data processing layer* has to provide a query engine that can be invoked by user applications.

It has as main task the search of data of interest using information retrieval facilities by means of proper information filters, as an example:

- query by keyword or tags and using specific metadata of the annotation schemata,
- query by example for cultural items with images,
- ontology based and query expansion semantic retrieval,
- etc.

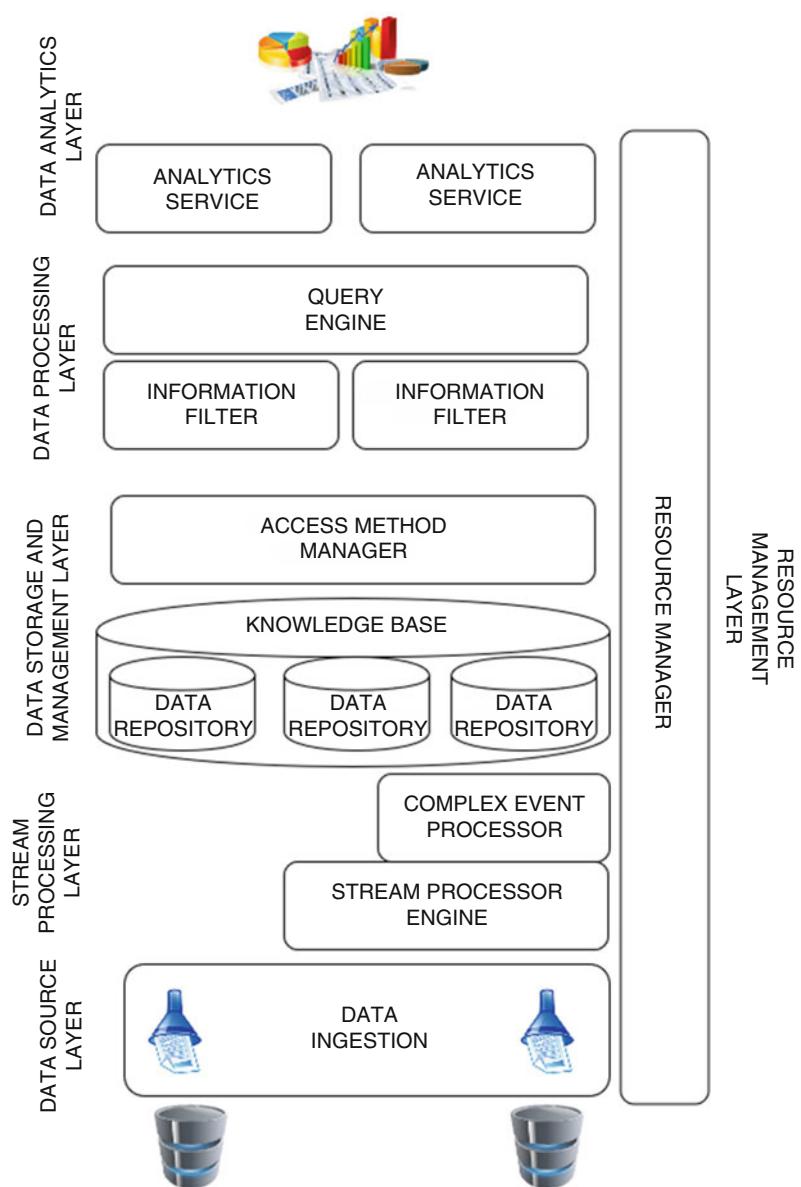
Finally, the *data analytics layer* is based on different analytics services allowing to create personalized “dashboards” for a given cultural environment. In addition, it provides basic data mining, graph analysis, and machine learning algorithms useful to infer new knowledge.

CHIS: A Big Data Platform for CH Applications

CHIS (cultural heritage information systems) (Castiglione et al. 2017) is the name of a big data platform which prototype was realized within

Big Data in Cultural Heritage, Fig. 1

Architecture of a big data platform for CH applications



the DATABENC project to support several CH applications.

It allows to query, browse, and analyze cultural digital contents from a set of distributed and heterogeneous digital repositories, providing several functionalities to guarantee CH preservation and fruition.

It shows all the functionalities and technical features required to a CH big data system. In the following, we describe the main characteristics, also providing some implementation details.

First of all, the CHIS data model is based on the concept of *cultural item*: examples are specific ruins of an archeological site, sculptures and/or pictures exhibited within a museum, historical buildings and famous squares in an old town center, and so on.

In the model, it is possible to describe a cultural item with respect to a variety of annotation schemata, by aggregating different harvesting sets of “metadata,” for example, the archeological view, the architectural perspective,

the archivist vision, the historical background, etc., as in the *Europeana Data Model* (EDM) (A proposal that aims at bridging the gaps between different harvesting standards providing a unique solution to metadata specification (<http://pro.europeana.eu/edm-documentation>)).

Stream processing of data from sensor networks has been realized according to a “publish-subscribe” paradigm; concerning its implementation details, CHIS uses *Apache Kafka* to realize message brokers and the publish-subscribe communication, *Apache Storm* as stream processing engine, and *Apache Zookeeper* as coordinator.

On the other hand, ingestion mechanisms from the other kinds of data sources (digital archives/libraries and social media networks) have been realized using the available APIs or by means of basic wrapping techniques.

All the extracted data are converted in a JSON format according to the described data model. In particular for multimedia contents, the descriptions in terms of metadata are captured and stored within CHIS, while raw data can be opportunely linked and, in other cases, temporarily imported into the system for content-based analysis (Bartolini and Patella 2015).

Data are then stored in the CHIS KB. In particular, the KB leverages a stack of different technologies that are typical of a big data system:

- The data describing basic properties of cultural items (e.g., name, short description, etc.) and basic information on users profiles are stored into a *key-value data store* (i.e., *Redis*).
- The complete description in terms of all the metadata using the different annotation schemata is in turn saved using a *wide-column data store* (i.e., *Cassandra*). A table for each kind of cultural objects is used, having a column for each “metadata family”; column values can be literals or URIs (according to LD paradigm).
- The *document store* technology (i.e., *MongoDB*) is used to deal with JSON messages, complete user profiles, and descriptions of internal resources (multimedia data and textual

documents, etc.) associated with a cultural item.

- All the relationships among cultural items within a cultural environment and interactions with users (behaviors) are managed by means of a *graph database* (i.e., *Titan*).
- The entire cartography related to a cultural environment together with PoIs is managed by a *GIS* (i.e., *PostGIS*);
- Multimedia data management has been realized using the *Windsurf* library (Bartolini and Patella 2015);
- CHIS exploits an *RDF store* (i.e., different *Allegrograph* instances) to memorize particular data views in terms of triples related to a given cultural environment and useful for specific aims, providing a SPARQL endpoint for the applications;
- All system configuration parameters, internal catalogues, and thesauri are stored in a relational database (i.e., *PostgreSQL*)
- All the basic analytics and query facilities are provided by *Spark* based on *HDFS*;
- Semantics of data can be specified by linking values of high-level metadata to some available internal (managed by *Sesame*) or external ontological schemata.

Eventually, several analytics can be performed on the stored data using graph analysis and machine learning library provided by *Spark*. As an example, the analysis of the graph coding interactions among users, multimedia objects, and cultural items can be properly used to support recommendation or influence analysis tasks.

This heterogeneous knowledge base provides basic *Restful* APIs to read/write data and further functionalities for importing/exporting data in the most common diffused web standards. CHIS adopts such heterogeneous technologies in order to meet the specific requirements of the applications dealing with the huge amount of data at stake.

For example, social networking applications typically benefit of graph database technologies because of their focus on data relationships.

In turn, more efficient technologies (key-value or wide-column stores) are required by tourism applications to quickly and easily access the data of interest.

The overall system architecture is made by using multiple virtual machines rent by a cloud provider, each equipped with *Hadoop/YARN* and *Spark* packages, accessing an horizontally scaling dynamic storage space that increases on demand, according to the evolving data memorization needs. It consists in a certain number of slave virtual machines (that can be increased over time in the presence of changing demands) and two additional ones that run as masters: one for controlling HDFS and another for resource management.

The CHIS platform has been then used to support several CH applications, as an example the multimedia guide for the *Paestum archeological site* described in Colace et al. (2015).

Concluding Remarks

This entry focused on the importance of big data technologies to support smart cultural environments. We outlined the main characteristics and functionalities required to a general big data platform for CH information management and described CHIS, i.e., a big data system to query, browse, and analyze cultural digital contents from a set of distributed and heterogeneous digital repositories, thus providing several functionalities to guarantee CH preservation and fruition.

References

- Albanese M, d'Acierno A, Moscato V, Persia F, Picariello A (2013) A multimedia recommender system. *ACM Trans Internet Technol (TOIT)* 13(1):3
- Amato F, Chianese A, Moscato V, Picariello A, Sperli G (2012) Snops: a smart environment for cultural heritage applications. In: Proceedings of the twelfth international workshop on Web information and data management. ACM, pp 49–56
- Bartolini I, Patella M (2015) Multimedia Queries in Digital Libraries. In: Colace F, De Santo M., Moscato V.,

Picariello A., Schreiber F., Tanca L. (eds) *Data Management in Pervasive Systems. Data-Centric Systems and Applications*. pp 311–325. Ed. Springer Cham Print. ISBN 978-3-319-20061-3.

Castiglione A, Colace F, Moscato V, Palmieri F (2017) Chis: a big data infrastructure to manage digital cultural items. *Futur Gener Comput Syst*. <https://doi.org/10.1016/j.future.2017.04.006>

Chen CP, Zhang CY (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inf Sci* 275:314–347

Colace F, De Santo M, Moscato V, Picariello A, Schreiber F, Tanca L (2015) Patch: a portable context-aware atlas for browsing cultural heritage. In: *Data management in pervasive systems*. Springer, Cham, pp 345–361

Cugola C, Margara A (2015) The complex event processing paradigm. In: Colace F., De Santo M., Moscato V., Picariello A., Schreiber F., Tanca L. (eds) *Data Management in Pervasive Systems. Data-Centric Systems and Applications*. Ed. Springer Cham Print. pp 113–133. ISBN 978-3-319-20061-3.

Kabassi K (2013) Personalisation Systems for Cultural Tourism. In: Tsihrintzis G., Virvou M., Jain L. (eds) *Multimedia Services in Intelligent Environments. Smart Innovation, Systems and Technologies*, vol 25. pp. 101–111. Ed. Springer, Heidelberg. ISBN 978-3-319-00374-0.

Panigati E, Schreiber FA, Zaniolo C (2015) Data streams and data stream management systems and languages. In: Colace F., De Santo M., Moscato V., Picariello A., Schreiber F., Tanca L. (eds) *Data Management in Pervasive Systems. Data-Centric Systems and Applications*. pp 93–111. Ed. Springer Cham Print. ISBN 978-3-319-20061-3.

Schaffers H, Komninos N, Pallot M, Trouse B, Nilsson M, Oliveira A (2011) Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation. In: Domingue J. et al. (eds) *The Future Internet. FIA 2011. Lecture Notes in Computer Science*, vol. 6656. pp 431–446. Ed. Springer, Berlin, Heidelberg. ISBN 978-3-642-20897-3.

Big Data in Mobile Networks

Pierdomenico Fiadino and Marc Torrent-Moreno
Data Science and Big Data Analytics Unit,
EURECAT (Technology Centre of Catalonia),
Barcelona, Spain

Synonyms

[Big Data in cellular networks](#)

Definitions

The term **Big Data in mobile networks** refers to datasets generated by and collected from mobile cellular networks. Such datasets consist in both the actual user data transported by the network (e.g., voice or data traffic) and metadata and control messages supporting network management.

Overview

A mobile or cellular network is a telecommunication system characterized by a radio last mile and the capability of managing the mobility of end terminals in a seamless fashion (i.e., without interrupting the communication). Such a network is geographically distributed over area units called cells (from which the name *cellular* originates), each served by one or more radio transceivers called *base stations*.

Mobile networks are nowadays the most popular mean for accessing both the traditional public switched telephone network (PSTN) and the public Internet. According to a projection by GSMA, the trade association of mobile operators, 70% of people worldwide will be mobile subscribers by the end of 2017 (Iji 2017). Considering the high penetration of cellular devices, data practitioners look at mobile technologies as an unprecedented information source. Indeed, every terminal produces a large amount of meta-information that can be exploited not only for network optimization and troubleshooting tasks but also for less technical-oriented use cases, such as the study of aggregated human behaviors and trends.

The data volume transferred over an operational network, and the associated metadata such as billing records, deeply varies depending on the network size (number of customers, links, geographical distribution, etc.) and monitored interfaces and can reach rates of several tens of terabytes per day. The analysis of such massive amount of data requires specific system specifications in terms of scalability for storage and processing and ability to deal with historical and real-time data analytics, depending on the

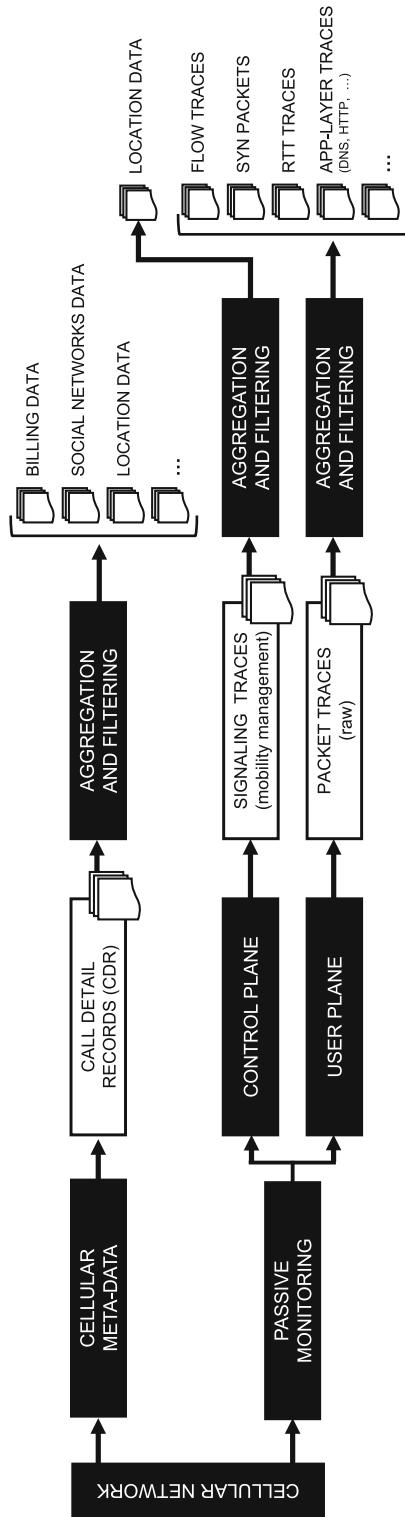
application field. Luckily, the progress in the field of Big Data analytics has facilitated the design and deployment of platforms for supporting Network Traffic Monitoring and Analysis (NTMA) applications in cellular context that meet these requirements. In particular, distributed frameworks based on the MapReduce paradigm have been recently started to be adopted in the field of NTMA, cfr. (Fontugne et al. 2014). Specifically, network monitoring approaches based on Apache Hadoop have been proposed in Lee and Lee (2012), while Liu et al. (2014) focus on rolling traffic analysis.

Cellular Data Types

Operational mobile networks generate an enormous amount of data, both on the **user plane** (e.g., user traffic transferred through the packet-switched domain or voice traffic through the circuit switched) and on the **control plane** (e.g., operators' billing data, signaling for terminals' mobility management and paging), as shown in He et al. (2016). In general, one can distinguish between two macro-groups of cellular data: **billing records** (also called CDR, inherently control data) and **passive traces** (which encompass both user and control data). Figure 1 shows a summary of the cellular data types.

Call Detail Record (CDR)

Call detail record (CDR) is the most extensively explored cellular data type. CDRs are metadata generated for supporting the billing of mobile subscribers. They consist in summary tickets of telephone transactions, including the type of activity (voice call, SMS, 2G/3G/4G data connection), the user(s) involved (the *caller* and the *callee*), a time stamp, technical details such as routing information, and the identifier of the cell offering connectivity to the terminal. The latter is especially interesting as it allows to localize the associated action in the boundaries of the cell's coverage area. In other words, CDRs carry details about subscribers' positions when they perform actions, aside from the details needed by the operator for charging them.



Big Data in Mobile Networks, Fig. 1 Summary of data types produced (or carried) by cellular networks. Billing metadata such as call detail records (CDRs) can be exploited for generating users' location data, social network information (from users' interactions), etc. Passively monitored traffic flowing in selected links allows the extraction of signaling messages of the cellular mobility management (which, in turn, provide fine-grained location and mobility data) and integral or partial user frames (packet traces) that can be further parsed to produce aggregated flow traces or filtered to application-specific traces

CDRs might not be as rich as other cellular data (e.g., signaling data including handover information), but they have been a popular study subject since the 2000s, in particular as a mean to observe human mobility and social networks at scale. The reason behind this interest should be sought in their rather high availability, as network operators are collecting these logs for billing which makes them a ready-made data source. They, in fact, do not require specialized infrastructure for the acquisition. Indeed, there is a fairly rich literature on the their exploitation for a plethora of applications, ranging from road traffic congestion detection and public transport optimization to demographic studies. Despite the initial interest, the research community gradually abandoned this data source as it has become clear that the location details conveyed by CDRs were biased by the users' activity degree. Specifically, the position of a user is observable in conjunction with an activity, which translates in the impossibility of locating a user with fine temporal granularity.

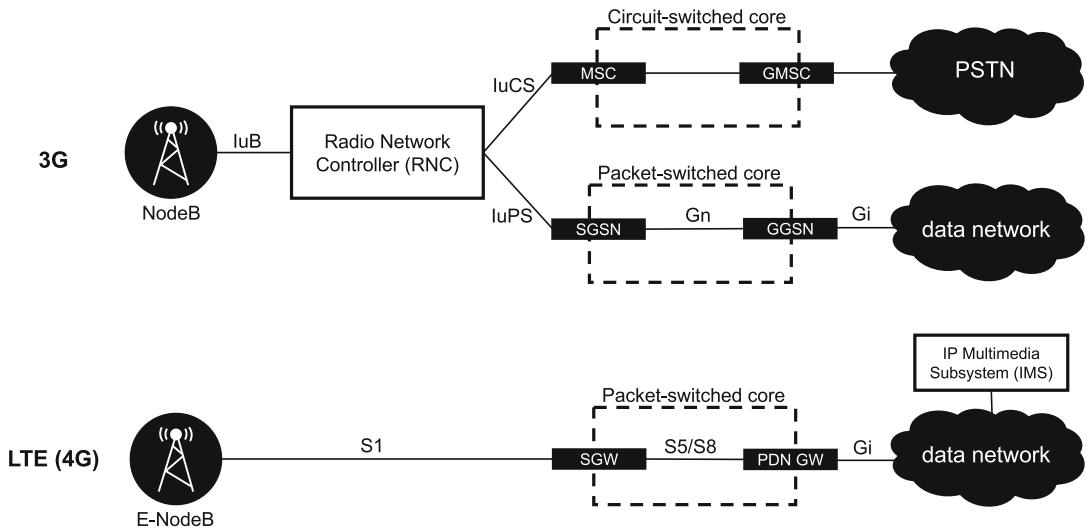
More recent studies (Fiadino et al. 2017), however, argue that the spreading of flat rates for voice and data traffic encourages users to generate more actions. In particular, competitive data plans are nowadays characterized by very high data volume limits, letting customers keep their terminals *always connected*. As a side effect, the quality of mobility data provided by CDRs improved as the average number of tickets per user has increased. A resurgence of research work tackling the study of CDRs is expected to happen in the next years. Indeed, the research community is already targeting some open issues, such as the lack of CDR standardization across operators and more accurate positioning within the cells' coverage areas (Ricciato et al. 2017).

Passive Monitoring

Passive traces are network logs collected through built-in logging capabilities of network elements (e.g., routers) or by wiretapping links through specialized acquisition hardware, such as data acquisition and generation (DAG) cards.

Nowadays, high-performance DAG cards are commercially available and are largely employed by cellular network operators in order to monitor their infrastructures at different vantage points (VPs). Passive measurements allow to gain network statistics at a large scale, potentially from millions of connected devices, hence with a high degree of statistical significance. This data type is intrinsically more powerful than CDRs: passive traces are exact copy of the traffic passing through a VP and open the possibility to fully reconstruct the activity of a mobile terminal through the study of both user traffic and control information. Needless to say, this poses serious challenges, in particular for what concern customers' privacy and the potential disclosure of business sensitive information. These issues have heavily limited the access to this kind of data to third parties and the research community.

The employment of passively collected traces for the purpose of network monitoring and management dates back to the 2000s: a first formalization of the techniques for passively monitoring mobile networks can be found in Ricciato (2006), where the author focuses on 3G networks and list common operator-oriented uses of the passively sniffed data. Despite the evolution of cellular networks (e.g., the passage from 3G to 4G standards), the main concept of passive monitoring has not changed much. Figure 2 illustrates a simplified cellular network architecture and highlights the links that can be wiretapped in order to collect passive traces. The cellular infrastructure is composed of two main building blocks: a radio access network (RAN), which manages the radio last mile, and a core network (CN). Fourth-generation networks (such as LTE) are characterized by less elements at the RAN, where the base stations (evolved NodeBs) are connected in a fully meshed fashion. The CN of 2G (GSM/EDGE) and 3G (UMTS/HSPA) networks is formed by the circuit-switched (CS) and the packet-switched (PS) domains, responsible for the traditional voice traffic and packet data traffic, respectively. In contrast, 4G networks are purely packet switched, as all type of traffic is carried over IP, including



Big Data in Mobile Networks, Fig. 2 A simplified representation of the 3G and 4G cellular network architecture. 3G networks are characterized by the presence of two cores (the packet switched, handling data connections, and

the circuit switched, handling traditional voice traffic). 4G networks have less network elements on the access network side and have just a packet-switched core, optimized for data connections

traditional voice calls which are managed by an IP multimedia subsystem (IMS).

In order to be reachable, mobile terminals notify the CN whenever they change location in the network topology (in other words, to which base stations they are attached to). Therefore, monitoring S1 (in LTE) or IuB/IuCS/IuPS interfaces (in 3G) would allow the collection of mobility information through the observation of network signaling (control) messages (i.e., **signaling traces**). Similarly to CDR datasets, an application for this type of traces is the study of human mobility, as specific signaling messages (such as *handovers*, *location area updates*, *periodic updates*, etc.) allow to reconstruct the trajectory of a terminal in the network topology, which, in turn, can be exploited to infer the actual mobility of the mobile customer with a certain degree of accuracy. These data could potentially provide a higher time granularity in the positioning of terminals w.r.t. CDRs, as the terminal's position is not necessarily observable in conjunction with an action (a call, SMS, data connection). The actual time granularity and position accuracy depend on multiple factors: activity state of the terminal (in general, *active* terminals provide their location changes with higher temporal granularity, even

when monitoring at the CN), monitored interfaces (the closer the passive sniffing occurred to the RAN, the higher the chance to get finer-grained positioning in the cellular topology), and use of triangulation (to calculate the position within the cell's coverage area, in case the signal strength is available). The interested reader might refer to Ricciato et al. (2015) for a comprehensive discussion on the passively observed location messages, their time and space granularity, and how they compare with the more popular CDRs.

Monitoring interfaces at the core of the PS domain, such as the Gn/Gi in 3G network and S5/S8/Gi interfaces in LTE, allows the collection of large-scale **packet traces**, i.e., a copy of frames at IP level augmented with metadata such as time stamps, the ID of the link where the wiretapping occurred, and a user (IMSI) and device (IMEI) identifier. The packet traces consist of the full frames (in case the payload is kept) or the header only. Cellular packet traces are extremely informative data, as they provide a complete snapshot of what happened in the network, supporting network management and troubleshooting (e.g., detection of network impairments, data-driven and informed dimensioning of resources, etc.) but also large-scale user

behavior studies (e.g., web surfing profiles, marketing analysis, etc.). For most applications, the header brings enough information, and therefore the payload can be discarded, making privacy and storage management easier. It should be noted that the payload is more and more often encrypted.

Raw packet traces can be further processed to produce more refined datasets. For example, by applying a *stateful* tracking of packets, it is possible to reconstruct **flow traces**, i.e., records summarizing *connections* (sequence of packets) between a source and a destination, aggregating by IPs and ports. Such logs could support, among others, the observation of throughput, latency, round-trip time (RTT), or other performance indicators that can be exploited for service-level agreement (SLA) verification and quality of service (QoS) studies. Another typical post-processing approach is filtering specific packet types (e.g., TCP SYN packets for the study of congestions over links and network security enforcement) or applications (i.e., **application-layer traces**, such as DNS, HTTP).

Uses of Cellular Data

In the last years, cellular network data has been broadly employed in different application areas, not necessarily with technical purposes. Indeed, the progress in the development of Big Data technologies, coupled with the decrease of prices for storage, has unlocked endless possibilities for the exploitation of the massive amount of data carried and produced by mobile networks.

In general, one can distinguish between two classes of use cases, i.e., **network-related** (tech-

nical) and **human-related** (nontechnical). The use cases in the first class aim at achieving *informed* network management and optimization actions, including more specific tasks such as detection of anomalies, troubleshooting, resource dimensioning, and security monitoring. Nontechnical applications focus on the study of human behavior relying on both user and control data exchanged among user equipments, cellular network components, and remote services. Some common human-related uses of cellular data are the study of human mobility (through CDRs or signaling messages), analysis of social networks, and web surfing behavior (through passively monitoring DNS or HTTP traffic at the CN), with applications in the field of marketing and behavioral studies.

The remainder of this section provide a non-exhaustive list of common uses of cellular data found in the scientific literature (also summarized in Table 1).

Technical Areas

The analysis of cellular data, and passive traces in particular, could support technical tasks aimed at achieving data-driven network engineering and optimization. There is an interesting research branch in this direction, targeting self-optimization, self-configuration, and self-healing based on machine learning approaches. One example is Baldo et al. (2014), where authors investigate the advantages of adopting Big Data techniques for **Self Optimized Networks** (SON) for 4G and future 5G networks. A comprehensive survey on the use of Big Data technologies and machine learning techniques in this field was edited by Klaine et al. (2017).

Big Data in Mobile Networks, Table 1 A non-exhaustive list of applications for mobile network data divided into two main categories (technical and nontechnical) plus a hybrid class of uses

Technical areas	Nontechnical areas	Hybrid areas
Anomaly detection	Churn prediction	Quality of Experience (QoE)
Traffic classification	Customer loyalty	
Network optimization	Web surfing	
Study of remote services	Marketing and tariff design	
Performance assessment and QoS	Human mobility	

Cellular passive traces can be also employed for studying the functioning and behavior of **remote Over The Top (OTT) Internet services** (Fiadino et al. 2016). Indeed, the popularity of mobile applications (e.g., video streaming, instant messaging, online social networks, etc.) poses serious challenges to operators as they heavily load the access of cellular networks. Hence, understanding their usage patterns, content location, addressing dynamics, etc. is crucial for better adapting and managing the networks but also to analyze and track the evolution of these popular services.

Congestions and Dimensioning

Nowadays, most of the services have moved over the IP/TCP protocol. The bandwidth-hungry nature of many mobile applications is posing serious challenges for the management and optimization of cellular network elements. In this scenario, monitoring congestions is critical to reveal the presence of under-provisioned resources.

The use of packet traces for the detection of bottlenecks is one of the earliest applications in the field of cellular passive monitoring. Ricciato et al. (2007) rely on the observation of both global traffic statistics, such data rate, as well as TCP-specific indicators, such as the observed retransmission frequency. Jaiswal et al. (2004) target the same goal by adopting a broader set of flow-level metrics, including TCP congestion window and end-to-end round-trip time (RTT) measurements. A more recent work that targets the early detection of bottlenecks can be found in Schiavone et al. (2014).

Traffic Classification

Network traffic classification (TC) is probably one of the most important tasks in the field of network management. Its goal is to allow operators to know what kind of services are been used by customers and how they are impacting the network by studying their passive packet traces. This is particularly relevant in cellular networks, where the resources are in general scarcer and taking informed network management decision is crucial.

The field of TC has been extensively studied: complete surveys can be found in Valenti et al. (2013) and Dainotti et al. (2012). Standard classification approaches rely on deep packet inspection (DPI) techniques, using pattern matching and statistical traffic analysis (Deri et al. 2014; Bujlow et al. 2015). In the last years, the most popular approach for TC is the application of machine learning (ML) techniques (Nguyen and Armitage 2008), including supervised and unsupervised algorithms.

The increasing usage of web-based services has also shifted the attention to the application of TC techniques to passive traces at higher levels of the protocol stack (specifically, HTTP). Two examples are Maier et al. (2009) and Erman et al. (2011), where authors use DPI techniques to analyze the usage of HTTP-based apps, showing that HTTP traffic highly dominates the total downstream traffic volume. In Casas et al. (2013a), authors characterize traffic captured in the core of cellular networks and classify the most popular services running on top of HTTP. Finally, the large-scale adoption of end-to-end encryption over HTTP has motivated the development of novel techniques to classify applications distributed on top of HTTPS traffic (Bermudez et al. 2012), relying on the analysis of DNS requests, which can be derived by filtering and parsing specific passive packet traces.

Anomaly Detection

Anomaly detection (AD) is a discipline aiming at triggering notifications when unexpected events occur. The definition of what is unexpected – i.e., what differs from a normal or predictable behavior – strictly depends on the context. In the case of mobile networks, it might be related to a number of scenarios, for example, degradation of performance, outages, sudden changes in traffic patterns, and even evidences of malicious traffic. All these events can be observed by mining anomalous patterns in passive traces at different level of granularities.

A comprehensive survey on AD for computer networks can be found in Chandola et al. (2009). An important breakthrough in the field of AD in large-scale networks was set by the work of

Lakhina et al. (2005), where authors introduced the application of the principal component analysis (PCA) technique to the detection of network anomalies in traffic matrices. Another powerful statistical approach tailored for cellular networks has been presented in D’Alconzo et al. (2010). The same statistical technique has been expanded and tested in a nationwide mobile network by Fiadino et al. (2014, 2015).

The study presented in Casas et al. (2016a) is particularly interesting: it applies clustering techniques on cellular traces to unveil device-specific anomalies, i.e., potentially harmful traffic patterns caused by well-defined classes of mobile terminals. The peculiarity of this kind of studies consists in the use of certain metadata (device brand, operating system, etc.) that are observable in passive traces collected in mobile networks (in general, monitoring fixed-line networks does not provide such degree of details on the subscribers).

User-Oriented Areas

Mobile network data implicitly bring large-scale information of human behavior. Indeed, operators can mine the satisfaction of users by observing their usage patterns. This type of analysis has clear applications in the fields of marketing, tariff design, and support business decisions in a data-driven fashion. An example is the **prediction of churning users** (i.e., customers lost to competitors): the study presented in Modani et al. (2013) relies on CDRs – and on the social network inferred from them – to extract specific behaviors that lead to termination of contracts.

Another area of user-oriented research is the study of the **quality of experience (QoE)**. The term QoE refers to a hybrid network- and human-related discipline that has the goal of estimating the level of satisfactions of customers when using IP-based services. In other words, practitioners in this field try to map network performance indicators (e.g., throughput, RTT, latency, etc.) with subjective scores assigned by users in controlled environments. Some examples of QoE studies conducted over cellular network data are Casas et al. (2013b, 2016b). In these works, authors study the QoE of popular applications accessed

through mobile devices from the perspective of passive cellular measurements, as well as subjective tests in controlled labs. The performance of multimedia services (VoIP, video streaming, etc.) over cellular networks is affected by a number of factors, depending on the radio access network (signal strength), core network setup (internal routing), and remote provisioning systems. This class of studies helps operators in properly dimensioning network resources not only to meet provisioning requirements from the network perspective but also taking into account the actual perception of quality by final users.

Human Mobility

The exploitation of mobile terminals and cellular networks as source of mobility information is possibly the most popular human-oriented topic in the field of mobile network data analysis. Cellular-based mobility studies have a high number of applications (e.g., city planning, analysis of vehicular traffic, public transportation design, etc.). Recently, cellular location data has been exploited to study the population density (Ricciato et al. 2015) and to infer socioeconomic indicators and study their relationship with mobility patterns (Pappalardo et al. 2016).

Mobility studies through cellular data are based on the assumption that the movements of a terminal in the context of the network topology reflect the actual trajectory of the mobile subscriber carrying the device. Location information can be extracted from CDRs or signaling messages devoted to the mobility management of terminals. Most of the existing literature is based on CDRs (Bar-Gera 2007; González et al. 2008; Song et al. 2010). However, since CDR datasets only log the position of users when an action occurs, their exploitation for the characterization of human mobility has been criticized (Ranjan et al. 2012). In Song et al. (2010), authors highlight some concerns in this direction and also show that users are inactive most of the time. The main limitation lies in the fact that the mobility perceived from the observation of CDRs is highly biased, as it strictly depends on the specific terminals’ action patterns. In other words, users are *visible*

during few punctual instants, which makes most of movements untraceable. Fiadino et al. (2017) claim that the situation is changing and nowadays CDRs are richer than the past, due to the increasing usage of data connections and background applications that has consequently increased the number of records. The now higher frequency of tickets allows a finer-grained tracking of users' positions. Some recent mobility studies, such as Callegari et al. (2017) and Jiang et al. (2017), are, in fact, successfully based on CDRs.

Few studies have explored more sophisticated approaches based on passively captured signaling messages – e.g., handover, location area updates, etc. Caceres et al. (2007). In Fiadino et al. (2012), authors studied the differences between CDRs and signaling data in terms of number of actions per user. Depending on the monitored interfaces, these approaches greatly vary in terms of cost and data quality (Ricciato 2006). Although the analysis of signaling is promising, there is a general lack of studies based on actual operational signaling data (some exceptions are the works by Fiadino et al. (2012) and Janecek et al. (2015), where authors focus on the study of vehicular traffic on highways and detection of congestions), as complex dedicated monitoring infrastructures for the extraction and immense data storage systems are required.

Privacy Concerns

The collection and exploitation of cellular data might raise some concerns. Indeed, given the penetration of mobile devices and the modern usage patterns, cellular networks carry an extremely detailed ensemble of private information. As seen before, by observing cellular metadata or specific network traces, it is possible to reconstruct the mobility of a mobile subscriber and infer the list of visited places, learn the acquaintances and build social networks, and even study the web surfing habits.

Customers' activity is associated to unique identifiers, such as the IMSI (International Mobile Subscriber Identity) and the IMEI (International Mobile Equipment Identity). These IDs

make it possible to link all the collected logs to single users and, in turn, to their identity. To protect customers' privacy, the **anonymization** of such unique identifiers is a common practice adopted by operators before handing privacy-sensitive datasets to internal or third-party practitioners. Indeed, sharing network traces with external entities is critical in some applications, e.g., detection of attacks with federated approaches (Yurcik et al. 2008).

A typical anonymization technique consists in applying nonreversible hashing functions, such as MD5 or SHA-1, with secret salt values to IMSI and IMEI. Being these hash functions cryptographically safe, it is still possible to associate all logs and actions of a single user to an anonymous identifier, with limited risks of *collisions* (differently from other simpler approaches, such as truncation of the last digits of customer identifiers). In addition, their one-way nature prevents reconstructing the original IDs. There is a rather rich literature on advanced anonymization techniques: a recent example can be found in Mivule and Anderson (2015).

It should be noted that all the listed applications for cellular data target macroscopic behaviors, observing users' traces on the large scale. The cases in which a single user needs to be addressed are rare and restricted to specific uses (e.g., detection of malicious users) or following authorized legal warrants. Nevertheless, the industry and the research community are addressing such concerns. For example, Dittler et al. (2016) propose an approach for location management that protects users' location privacy without disrupting the basic functionalities of cellular networks. On the other hand, the increasing adoption of encryption protocols to transport user data (e.g., HTTPS, instant messaging and VoIP with end-to-end encryption, etc.) limits some potentially abusive uses of passive traces.

Cross-References

- ▶ [Big Data in Computer Network Monitoring](#)
- ▶ [Big Data in Network Anomaly Detection](#)

References

- Baldo N, Giupponi L, Mangues-Bafalluy J (2014) Big data empowered self organized networks. In: European wireless 2014; 20th European wireless conference, pp 1–8. <https://doi.org/10.5281/zenodo.268949>
- Bar-Gera H (2007) Evaluation of a cellular phone-based system for measurements of traffic speeds and travel times: a case study from Israel. *Transp Res Part C Emerg Technol* 15(6):380–391. <https://doi.org/10.1016/j.trc.2007.06.003>
- Bermudez IN, Mellia M, Munafó MM, Keralapura R, Nucci A (2012) DNS to the rescue: discerning content and services in a tangled web. In: Proceedings of the 2012 internet measurement conference, IMC'12, pp 413–426
- Bujlow T, Carela-Español V, Barlet-Ros P (2015) Independent comparison of popular DPI tools for traffic classification. *Comput Netw* 76:75–89. <https://doi.org/10.1016/j.comnet.2014.11.001>
- Caceres N, Wideberg JP, Benitez FG (2007) Deriving origin destination data from a mobile phone network. *IET Intell Transp Syst* 1(1):15–26. <https://doi.org/10.1049/iet-its:20060020>
- Callegari C, Garropo RG, Giordano S (2017) Inferring social information on foreign people from mobile traffic data. In: 2017 IEEE international conference on communications (ICC), pp 1–6. <https://doi.org/10.1109/ICC.2017.7997255>
- Casas P, Fiadino P, Bär A (2013a) Ip mining: extracting knowledge from the dynamics of the internet addressing space. In: Proceedings of the 2013 25th international teletraffic congress (ITC), pp 1–9. <https://doi.org/10.1109/ITC.2013.6662933>
- Casas P, Seufert M, Schatz R (2013b) Youqmon: a system for on-line monitoring of youtube QoE in operational 3G networks. *SIGMETRICS Perform Eval Rev* 41(2):44–46
- Casas P, Fiadino P, D’Alconzo A (2016a) When smartphones become the enemy: unveiling mobile apps anomalies through clustering techniques. In: Proceedings of the 5th workshop on all things cellular: operations, applications and challenges (ATC’16), pp 19–24
- Casas P, Seufert M, Wamser F, Gardlo B, Sackl A, Schatz R (2016b) Next to you: monitoring quality of experience in cellular networks from the end-devices. *IEEE Trans Netw Serv Manag* 13(2):181–196
- Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. *ACM Comput Surv* 41(3):15:1–15:58
- Dainotti A, Pescape A, Claffy K (2012) Issues and future directions in traffic classification. *IEEE Netw* 26(1):35–40. <https://doi.org/10.1109/MNET.2012.6135854>
- D’Alconzo A, Coluccia A, Romirer-Maierhofer P (2010) Distribution-based anomaly detection in 3G mobile networks: from theory to practice. *Int J Netw Manag* 20(5):245–269. <https://doi.org/10.1002/nem.747>
- Deri L, Martinelli M, Bujlow T, Cardigliano A (2014) nDPI: open-source high-speed deep packet inspection. In: 2014 international wireless communications and mobile computing conference (IWCMC), pp 617–622. <https://doi.org/10.1109/IWCMC.2014.6906427>
- Dittler T, Tschorisch F, Dietzel S, Scheuermann B (2016) ANOTEL: cellular networks with location privacy. In: 2016 IEEE 41st conference on local computer networks (LCN), pp 635–638. <https://doi.org/10.1109/LCN.2016.110>
- Erman J, Gerber A, Sen S (2011) HTTP in the home: it is not just about PCs. *SIGCOMM Comput Commun Rev* 41(1):90–95. <https://doi.org/10.1145/1925861.1925876>
- Fiadino P, Valerio D, Ricciato F, Hummel KA (2012) Steps towards the extraction of vehicular mobility patterns from 3G signaling data. Springer, Berlin/Heidelberg, pp 66–80
- Fiadino P, D’Alconzo A, Bär A, Finamore A, Casas P (2014) On the detection of network traffic anomalies in content delivery network services. In: 2014 26th international teletraffic congress (ITC), pp 1–9. <https://doi.org/10.1109/ITC.2014.6932930>
- Fiadino P, D’Alconzo A, Schiavone M, Casas P (2015) Rcatool – a framework for detecting and diagnosing anomalies in cellular networks. In: Proceedings of the 2015 27th international teletraffic congress (ITC’15), pp 194–202
- Fiadino P, Casas P, D’Alconzo A, Schiavone M, Baer A (2016) Grasping popular applications in cellular networks with big data analytics platforms. *IEEE Trans Netw Serv Manag* 13(3):681–695. <https://doi.org/10.1109/TNSM.2016.2558839>
- Fiadino P, Ponce-Lopez V, Antonio J, Torrent-Moreno M, D’Alconzo A (2017) Call detail records for human mobility studies: taking stock of the situation in the “always connected era”. In: Proceedings of the workshop on big data analytics and machine learning for data communication networks (Big-DAMA’17), pp 43–48
- Fontugne R, Mazel J, Fukuda K (2014) Hashdoop: a mapreduce framework for network anomaly detection. In: 2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS), pp 494–499. <https://doi.org/10.1109/INFOWKSHPS.2014.6849281>
- González MC, Hidalgo C, Barabási A (2008) Understanding individual human mobility patterns. *Nature* 453:779–782. <https://doi.org/10.1038/nature06958>, 0806.1256
- He Y, Yu FR, Zhao N, Yin H, Yao H, Qiu RC (2016) Big data analytics in mobile cellular networks. *IEEE Access* 4:1985–1996. <https://doi.org/10.1109/ACCESS.2016.2540520>
- Iji M (2017) GSMA intelligence – unique mobile subscribers to surpass 5 billion this year. <https://www.gsmaintelligence.com/research/2017/02/unique-mobile-subscribers-to-surpass-5-billion-this-year/613>. Accessed 08 Feb 2018
- Jaiswal S, Iannaccone G, Diot C, Kurose J, Towsley D (2004) Inferring TCP connection characteristics through passive measurements. In: IEEE INFOCOM 2004, vol 3, pp 1582–1592. <https://doi.org/10.1109/INFCOM.2004.1354571>

- Janecek A, Valerio D, Hummel K, Ricciato F, Hlavacs H (2015) The cellular network as a sensor: from mobile phone data to real-time road traffic monitoring. *IEEE Trans Intell Transp Syst.* <https://doi.org/10.1109/TITS.2015.2413215>
- Jiang S, Ferreira J, Gonzalez MC (2017) Activity-based human mobility patterns inferred from mobile phone data: a case study of Singapore. *IEEE Trans BigData.* <https://doi.org/10.1109/TBDA.2016.2631141>
- Klaine PV, Imran MA, Onireti O, Souza RD (2017) A survey of machine learning techniques applied to self organizing cellular networks. *IEEE Commun Surv Tutorials* 99:1–1. <https://doi.org/10.1109/COMST.2017.2727878>
- Lakhina A, Crovella M, Diot C (2005) Mining anomalies using traffic feature distributions. *SIGCOMM Comput Commun Rev* 35(4):217–228
- Lee Y, Lee Y (2012) Toward scalable internet traffic measurement and analysis with Hadoop. *SIGCOMM Comput Commun Rev* 43(1):5–13
- Liu J, Liu F, Ansari N (2014) Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop. *IEEE Netw* 28(4):32–39. <https://doi.org/10.1109/MNET.2014.6863129>
- Maier G, Feldmann A, Paxson V, Allman M (2009) On dominant characteristics of residential broadband internet traffic. In: Proceedings of the 9th ACM SIGCOMM conference on internet measurement conference (IMC'09). ACM, New York, pp 90–102. <https://doi.org/10.1145/1644893.1644904>
- Mivule K, Anderson B (2015) A study of usability-aware network trace anonymization. In: 2015 science and information conference (SAI), pp 1293–1304. <https://doi.org/10.1109/SAI.2015.7237310>
- Modani N, dey k, Gupta R, Godbole S (2013) CDR analysis based telco churn prediction and customer behavior insights: a case study. In: Lin X, Manolopoulos Y, Srivastava D, Huang G (eds) Web information systems engineering – WISE 2013. Springer, Berlin/Heidelberg, pp 256–269
- Nguyen TTT, Armitage G (2008) A survey of techniques for internet traffic classification using machine learning. *IEEE Commun Surv Tutorials* 10(4):56–76. <https://doi.org/10.1109/SURV.2008.080406>
- Pappalardo L, Vanhoof M, Gabrielli L, Smoreda Z, Pedreschi D, Giannotti F (2016) An analytical framework to nowcast well-being using mobile phone data. *Int J Data Sci Anal* 2:75–92
- Ranjan G, Zang H, Zhang Z, Bolot J (2012) Are call detail records biased for sampling human mobility? *SIGMOBILE Mob Comput Commun Rev* 16(3):33
- Ricciato F (2006) Traffic monitoring and analysis for the optimization of a 3G network. *Wireless Commun* 13(6):42–49
- Ricciato F, Vacirca F, Svoboda P (2007) Diagnosis of capacity bottlenecks via passive monitoring in 3G networks: an empirical analysis. *Comput Netw* 51(4):1205–1231
- Ricciato F, Widhalm P, Craglia M, Pantano F (2015) Estimating population density distribution from network-based mobile phone data. Publications Office of the European Union. <https://doi.org/10.2788/162414>
- Ricciato F, Widhalm P, Pantano F, Craglia M (2017) Beyond the “single-operator, CDR-only” paradigm: an interoperable framework for mobile phone network data analyses and population density estimation. *Pervasive Mob Comput* 35:65–82. <https://doi.org/10.1016/j.pmcj.2016.04.009>
- Schiavone M, Romirer-Maierhofer P, Ricciato F, Baiocchi A (2014) Towards bottleneck identification in cellular networks via passive TCP monitoring. In: Ad-hoc, mobile, and wireless networks – 13th international conference (ADHOC-NOW 2014). Proceedings, Benidorm, 22–27 June 2014, pp 72–85
- Song C, Qu Z, Blumm N, Barabási AL (2010) Limits of predictability in human mobility. <https://doi.org/10.1126/science.1177170>
- Valenti S, Rossi D, Dainotti A, Pescapè A, Finamore A, Mellia M (2013) Reviewing traffic classification. In: Biersack E, Callegari C, Matijasevic M (eds) Data traffic monitoring and analysis: from measurement, classification, and anomaly detection to quality of experience. Springer, Berlin/Heidelberg, pp 123–147
- Yurcik W, Woolam C, Hellings G, Khan L, Thuraisingham B (2008) Measuring anonymization privacy/analysis tradeoffs inherent to sharing network data. In: NOMS 2008 – 2008 IEEE network operations and management symposium, pp 991–994. <https://doi.org/10.1109/NOMS.2008.4575265>

B

Big Data in Network Anomaly Detection

Duc C. Le and Nur Zincir-Heywood
Dalhousie University, Halifax, NS, Canada

Synonyms

Network anomaly detection; Network normal traffic/Behavior modeling; Network outlier detection

Definitions

Network anomaly detection refers to the problem of finding anomalous patterns in network activities and behaviors, which deviate from normal network operational patterns. More specifically, in network anomaly detection context, a set of network actions, behaviors, or observations is

pronounced anomalous when it does not conform by some measures to a model of profiled network behaviors, which is mostly based on modelling benign network traffic.

Overview

In today's world, networks are growing fast and becoming more and more diverse, not only connecting people but also things. They account for a large proportion of the processing power, due to the trend of moving more and more of the computing and the data to the cloud systems. There might also come the time when the vast majority of things are controlled in a coordinated way over the network. This phenomenon not only opens many opportunities but also raises many challenges at a much larger scale.

Due to the ever-expanding nature of the network, more and more data and applications are being created, used, and stored. Furthermore, novel threats and operational/configuration problems start to arise. This in return creates the environment for systems such as the network anomaly detection to become one of the most prominent components in the modern networks.

Examples of anomalies on the network

Network anomalies are mostly categorized into two types: performance-related anomalies and security-related anomalies (Thottan and Ji 2003).

- *Anomaly in performance* This type of anomalies represents abrupt changes in network operations. Most of the time, they may be generated by configuration errors or unexpected shifting in the network demand or supply resources. Several examples of such anomalies include server or router failures and transient congestions. A server failure, such as a web server failure, could occur when there is a swift increase in the demand for the site's content to the point that it exceeds the capabilities of the servers. Network congestion at short time scales occurs due to hot spots in the network that may be caused by the

failures of network devices or connections or excessive traffic load at specific regions of the network. In some instances, network anomalies could be the result of errors in the network configuration, such as routing protocol configuration errors causing network instability.

- *Anomaly in network security* This is the main focus of many research works in the literature, where the network anomalous behaviors are caused by security-related problems. Examples are distributed denial-of-service (DDoS) attacks and network anomalies caused by intrusions or insider threats. DDoS attacks are conceivably one of the most noticeable types of anomalies, where the services offered by a network are disrupted. This effectively prevents normal user from accessing the content by overloading the network infrastructures or by disabling a vital service such as domain name server (DNS) lookups. In case of network intrusions, once the malicious entity obtained the control of some parts of a network, those network resources could end up being used for a large scheme DDoS attack or for other illegitimate resource-intensive tasks. On the other hand, similar to insider threats, the attackers could silently exploit the controlled resources to gather valuable information and credentials, such as credit cards and user ID and passwords, or organization's operational information and strategies. This could result in more resilient, hard to disclose types of anomalies, which in many cases may be overlooked by many anomaly detection systems.
- Another type of anomaly that may appear on the network is the shift or drift in the normal network behaviors. This is labelled as anomalous in many cases, because a deviation is seen compared to the modelled behaviors. Based on specific anomaly detection systems, these observations can be seen as informational alerts or just simply false positives.

Generally, in a specific work, only one type of anomaly is considered the main focus, but from network anomaly detection perspective in this chapter, all types are important and critical

to network operations. The correctly identified deviation from normal network behavior may indicate the troubles in the network configuration, unusual high network demands, connection failures, breached security, or attacks on the network. All of this information is crucial for the health of the network.

From the examples above, given the scale of today's networks, network anomaly detection easily becomes one typical example of big data applications (Laney 2001):

- *High volume* The network anomaly detection data can come from a wide range of sources. Network data collected at host level (from servers to personal devices to things) can be traffic log/capture, system data, or application/user data. At the network level, data can be collected from network probes, network traffic capturing devices, or management data (which can be routers, switches, firewalls) in different locations across the network. Given the scale of networks and the connected devices/systems on the Internet today, one can easily imagine the unprecedented amount of data generated by networks that needs to be analyzed each day. A medium-sized Internet service provider or any organization, such as a university, with a thousand users, can easily collect terabytes of data a day for analyzing. Furthermore, anomaly detection requires processing and modelling of network data over a sufficiently long period for detecting peculiar behaviors in new data.
- *Velocity* Network data is generated perpetually at high speed. Especially, the network anomaly detection problem requires the data to be processed quickly to deal with any possible arising problems within a timely manner.
- *Variety* As stated above, network anomaly detection data comes from a wide variety of sources, from numeric data summarizing network activities to structured time sequence data and unstructured data from mixed sources all over the network and systems. Moreover, variety also comes when the context of networks is expanded to include cellular net-

works or to cyber-physical systems such as power grids and transportation networks.

- *Variability* In addition to the increasing velocities and varieties of data, network data is notorious for including both stationary and nonstationary behaviors. The data stream is not only fluctuated by time but also constantly changing based on the applications and services users employ. This can be challenging to model, understand, and manage.
- *Complexity* Network anomaly detection is a complex task, firstly because analyzing network data itself is hard in nature. Network data comes from multiple sources and at different levels of the network protocol stack. This in return makes it difficult to link, match, cleanse, and transform the data across systems. Besides, most of the network traffic is encrypted to some extent, which inhibits direct analysis of the content. Secondly, anomalies usually are rare events that can easily be overlooked in the mountain of network data gathered a day, making the task "looking for a needle in a haystack." Finally, the task of finding network anomalies is also vague, because anomalies are basically not defined on its own but by a deviation from what is modelled to be the normal behavior for a given system/device.

Literature Review in Network Anomaly Detection

Network anomaly detection has a vibrant development history connected to the growth of networks and connected systems. There have been a good number of techniques for network anomaly detection developed by researchers. In this section, a broad classification of network anomaly detection methods based on the use of big data/machine learning applications is adopted for summarizing anomaly detection techniques.

Early research focused on applying predefined rules and patterns or statistical tools for the task (Thottan and Ji 2003; Bhuyan et al. 2014).

- Rule-based and pattern matching approaches mainly relied on expert systems encoded in a set of rules and patterns for matching against incoming network traffic and behaviors. Examples of this category are Snort and Bro, where anomaly detection can be done in rules or scripts to identify excessive amount in specific properties, such as utilization of bandwidth, the number of open transmission control protocol connections, or the number of Internet control message protocol packets, which exceeds a predefined threshold.
- Ontology and logic-based approaches attempt to model network behaviors using expressive logic structures, such as finite state machines and ontologies for encoding knowledge of experts in the domain. In those models, the sequence of alarms and actions obtained from different locations on a given network are modelled by a reasoning model designed by the experts not only for detecting anomalies and known intrusions but also describing and diagnosing the related problems. Notable works in this category are by Ilgun et al. (1995) and Shabtai et al. (2010), which are based on the state machine model and the knowledge-based temporal abstraction, respectively.

The efficiency of the above approaches quintessentially depends on the accuracy of expert knowledge embedded in the defined rules, patterns, and ontologies. Thus, it is possible that the constructed models do not adapt adequately to the evolution in the network and cybersecurity environments, allowing new faults and threats evade detection. Moreover, given a new network, it may be necessary to spend a considerable amount of time to model the needs for specific patterns/rules for building traffic models to match network conditions. As network topologies and traffic conditions are expanding and evolving, the techniques based on traffic inspection and rule/pattern/model matching may not scale gracefully.

Envisioning the scale and variability of networks and their potential growth in the near future, one can easily realize the need for more

efficient and easily adaptable tools for anomaly detection. Thus, machine learning techniques naturally found applications in this field for their ability to automatically learn from the data and extract patterns that can be used for identifying anomalies in a timely manner. Although most of the machine learning applications in cybersecurity generally and intrusion detection specifically are based on supervised learning techniques, in anomaly detection, unsupervised learning is the most applied technique due to the ability of learning without the label information, i.e., a priori knowledge of anomalies, which essentially is the defining characteristic of anomaly detection.

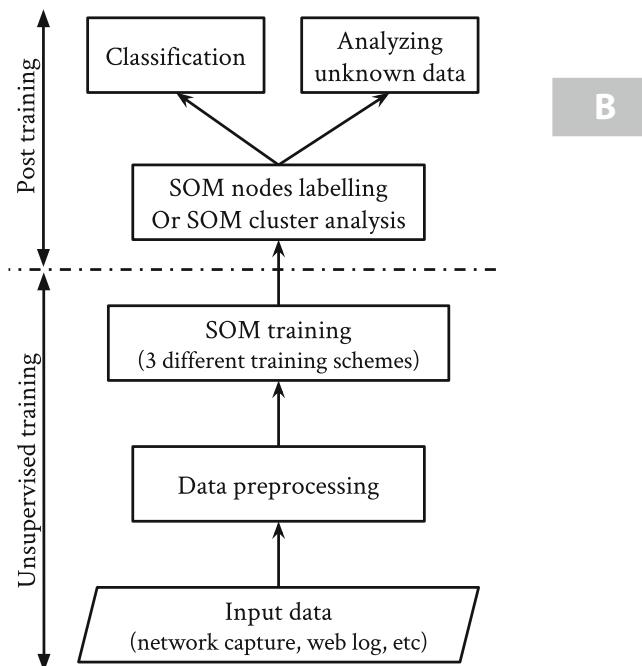
- Unsupervised learning techniques can be used to construct a model generalizing normal network and user behaviors and producing measures which could be used to detect anomalies in network traffic. Examples of different works in this area are by Dainotti et al. (2007, 2009), where temporal correlation, wavelet analysis, and traditional change point detection approaches are applied to produce a network signal model and worm traffic model, and (Sequeira and Zaki 2002) and (Rashid et al. 2016), where user profiles are created from the sequence of user actions in a time window using clustering techniques and hidden Markov model.
- A more direct approach using unsupervised learning for anomaly detection is applying clustering analysis and outlier detection methods. In this approach, clustering algorithms are used to find significant clusters representing majority of normal network/system behaviors. From clustering results, anomalies can be detected automatically as the data instances that do not fit into the constructed clusters of normal data using outlier detection methods. Notable works are Otey et al. (2006), Jiang et al. (2006), Finamore et al. (2011) and Casas et al. (2012).
- Unsupervised learning results can be analyzed by human experts with or without the use of visualization and other supervised learning methods to give more insight. Notable works are Kayacik et al. (2007), Perdisci et al.

(2006) and Leung and Leckie (2005). The self-organizing map (SOM) is a well-known unsupervised learning technique with the ability of not only summarizing the data learned in a topologically preserved way but also presenting the data visually for further inspection (Kayacik et al. 2007; Le et al. 2016; Rajashekhar et al. 2016). Recently, Veeramachaneni et al. (2016) brings the concept of big data closer to anomaly detection, where human experts, multiple outlier detection techniques, and supervised learning are combined to learn from a large amount of data to detect network anomalies and intrusions.

Key Research Findings on Network Anomaly Detection

This section describes a framework based on unsupervised machine learning for network data analysis and is applicable to network anomaly detection (Fig. 1). Following the progress of research described above, one can easily notice one trend in network anomaly detection research. That is to apply machine learning tools with strong pattern recognition and visualization abilities not only to learn from the data and extract results but also to incorporate the domain expert's knowledge and support network administrators. Having a unique combination of unsupervised learning and topology preserving visualization capabilities (e.g., producing hit maps, U-matrix), self-organizing map (Kohonen 2001) is applied in the framework as the underlying technique to support network data analysis in a wide range of conditions.

Fundamentally the framework is based on a *data-driven* approach using unsupervised learning with visualization abilities to learn a wide range of network and system behaviors with minimum prior knowledge. The data is extracted to numerical vectors; examples are network flows from traffic captures or access request characteristics for web log files. Subsequently, the data is preprocessed, for normalization purposes or for dimensionality reduction purposes, before being input to the SOM learning algorithm. Three dif-



Big Data in Network Anomaly Detection, Fig. 1
Proposed framework architecture

ferent schemes based on different training data compositions were tested in the framework: (i) use both known legitimate (normal) and known malicious data for training purposes, as done in the previous supervised learning approaches; (ii) use only legitimate data, as done in the previous anomaly detection methods; and (iii) use only malicious data, as done in some of the previous one-class classifier approaches. The reason behind these training schemes is to not only represent the real-life data acquisition conditions but also to shed light into understanding the performance gains/losses under different types/amounts of labelling information, i.e., ground truth. For example, the data collected by honeypots is usually only considered as being representative of malicious behavior. On the other hand, in idealistic cases of networks where there are no previous attacks, the data collected contains only legitimate traffic. Moreover, even when a threat is discovered in the collected traffic, it is very challenging to fully identify the extent of the threat and label the data collected.

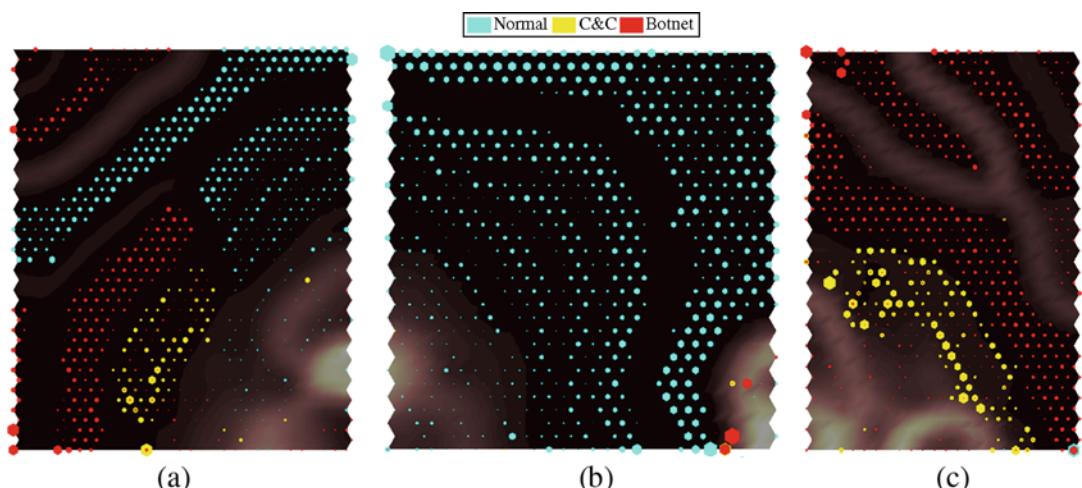
The SOM is trained in an unsupervised manner, with the assumption that learning solely from data, it could learn well-separated clusters to differentiate distinct communication behaviors. Post training, to quantify the system performance, if the ground truth of the training data is available, can be used for labelling the SOM nodes. On the other hand, when label information is not available, cluster analysis based on the trained SOM topology visualization can be used along with expert knowledge and information from other sources to derive meaningful insights from the data. The SOM-based framework then can be used to analyze incoming network data and support network administrators in operation by visualizing the traffic.

Examples of applications The framework can be applied in many network situations with different sources of data and domain knowledge incorporated. Some applications are presented in detail in Le et al. (2016) and Le (2017). In this chapter, a brief description of one example in applying the framework for network flow data analysis in order to detect Virut botnet in a mixed dataset from García et al. (2014) is presented.

Network flows are the summary of connections between hosts, which include descriptive statistics that are calculated from aggregating

Network and Transport layers' header information of the transmitted packets and can be independent from the application layer data. Given that network traffic encryption is very popular in both benign applications for protecting users' privacy and sensitive information, and malicious applications for hiding from the detection systems analyze network packet payload, the detection approach using only netflows exported from packet headers may improve the state of the art in unsupervised learning-based data analysis. Furthermore, the use of network flows can significantly reduce the computational requirement of a network monitoring system by allowing the analysis of each flow as a summary of packets and avoiding deep packet inspection.

Figure 2 presents the results obtained using the three training schemes in the framework. It is clear that the first training scheme, where SOM is formed using both known normal and malicious data, was able to form well-separated regions for the classes and achieve 99.9% detection rate with only 0.6% false-positive rate. Similarly, the second training scheme, which is the most suitable for network anomaly detection applications where only normal data is used to form the SOM, achieved promising results at 80.9% detection rate and 6% false-positive rate. Figure 2b shows that SOM trained using scheme (ii) was able to



Big Data in Network Anomaly Detection, Fig. 2 SOM visualization of test data in the Virut dataset based on the three training schemes. (a) scheme (i) (b) scheme (ii) (c) scheme (iii)

identify the malicious data by mapping them to an unpopular region, which is distanced from the training data. On the other hand, scheme (iii) SOM was able to differentiate only 3.4% of the normal data from the malicious data used for training (Fig. 2c). The visualization capabilities supported by the framework, as shown in Fig. 2, could give network administrators options to quickly analyze network data, even without ground truth by accessing the regions formed by the SOM. In-depth analysis and details of applications of the framework in other use cases, such as web attack and novel botnet detection, can be found in Le (2017).

Challenges in Applying Big Data in Network Anomaly Detection

Although machine learning has been extensively applied in the task of anomaly detection, there are certain challenges that need to be overcome for a successful realization of research results in production network environments. First and foremost, there are challenges which come from the nature of the anomaly detection task that all the approaches, including machine learning-based approaches, must address. Some of them are already summarized in Sommer and Paxson (2010).

- The problems related to the data:
 - Network data is diverse and abundant, yet obtaining high-quality data for designing and evaluating anomaly detection systems typically involves considerable difficulties. Many companies and organizations are prevented from sharing data by agreements protecting users' identities and privacy. Even when the data is shared, most of it comes without any form of ground truth for training and evaluating anomaly detection systems with confidence.
 - Secondly, network anomaly detection data is highly skewed; most of the time, only a small unidentified portion of data is anomalous. One example is insider threats,

where the attackers can perform malicious actions over a long duration, sometimes months (or longer) to evade monitoring systems. Hence detection systems and network operations team could fail to recognize the small changes as the sign of anomalies.

- Network data can be encrypted and/or anonymized, which may inhibit many detection systems that monitor network traffic.
- In many cases, the developers may face the dilemma of impurity in training data. This can seriously affect the efficiency of anomaly detection systems, as a model inadvertently built with abnormal data would encode that as a normal behavior, which makes it incapable to detect future anomalies of the type.
- The very nature of anomaly detection may itself pose a problem to developers in defining an anomaly and in distinguishing it from normal network behaviors. Similarly, machine learning techniques may have trouble identifying the novel behaviors that are dissimilar from learned patterns for anomaly detection tasks. This is unlike other applications of machine learning, such as classification and recommendation systems, where the main focus is in finding similarity.
- There are significant challenges in generalizing anomaly detection methods to become independent of the environment. Different network environments usually have different requirements and conditions for defining normal and anomaly behaviors. Moreover, in each network the concepts of anomaly and normal profiles are continuously shifting or drifting. Therefore, it is necessary to study dynamic, adaptive, and self-evolving anomaly detection systems to overcome limitations of the traditional static models.
- Ideally, an anomaly detection method should pursue highest anomaly detection rates while avoiding a high rate of false alarms. However, in designing a novel anomaly detection method, one always has to take into account a trade-off between the two measures.

Considering the dominance of normal data and the scale of networks, even a small false-positive rate may result in a catastrophic amount of false alarms that need attention from human experts – network operations team. Lessons have been learned that the detection systems with low precision would be impractical for production network environments. This requires the anomaly detection system not only to have the ability to detect abnormal network behaviors but also to correlate and to learn from the alarms in order to provide meaningful feedback to/from human experts to improve.

To apply big data successfully in the anomaly detection problem, not only the aforementioned challenges but also the greater scope and scale of challenges need to be addressed and overcome. These include but are not limited to the following aspects of network-based systems:

- Runtime limitation presents an important challenge for a network anomaly detection system, especially when big data is applied to deal with larger networks. Reducing computational complexity in preprocessing, training, and deployment of such systems is a priority.
- Data for network anomaly detection systems is presented at multiple levels of granularity and formats. The data can be acquired at host or network level, from end devices, network devices, security devices, or servers, in different formats and structures. In small networks, the data can be processed manually by experts; however, when it comes to big data applications, the problem of data acquisition, data representation, and preprocessing must be addressed systematically in order to provide high-quality data for training big data algorithms efficiently. Similarly, developing appropriate and fast feature selection methods is crucial not only to ensure efficiency of anomaly detection systems but also to provide compact representation of the data with sufficient information.

- The very nature of network is streaming. In small networks, gradual drifting and shifting of behaviors and concepts in data can be addressed by retraining learned data models periodically. However, in big data applications, the prevalent phenomenon needs to be addressed adequately from a system design perspective. This requires dynamic and adaptive learning algorithms and self-evolving architectures that are capable of working on high-speed data streams. Furthermore, such systems need to have capabilities for revising/updating themselves in a timely manner according to the ever-changing dynamics of the network environment without sacrificing from their detection performances.
- In big data environments, there is a significant challenge in converting anomaly warnings to meaningful alerts and producing patterns for future prediction, considering the tremendous amount of data that a warning system must process a day. This will require big data systems that are able to cooperate with human experts and continuously learn from human input to evolve itself to adapt to the new normal and malicious behaviors on the network.

Conclusion

Network anomaly detection has become more and more prevalent and crucial to monitor and secure networks. Given the scale and dynamics of a network, developing practical big data applications for the task is an open field of research. Due to the very nature of network data and learning algorithms, many traditional approaches for network analytics and anomaly detection are not scalable. Moreover, novel and task-specific challenges need to be addressed earnestly in order to successfully introduce new big data applications to the field.

Future directions for research In light of the challenges discussed above, we aim to help to strengthen the future research on big data applications for anomaly detection systems by a concrete set of recommendations:

- Drift and shift in network data streams can be addressed using active online learning algorithms (Khanchi et al. 2017).
- Dynamic anomaly detection systems should be capable of learning from multiple sources such as the offline data sources, streaming data sources, expert knowledge, and feedbacks. It has been shown in Le et al. (2016) that learning algorithms with visualization capabilities not only achieve the goals of anomaly detection via clustering and outlier detection techniques but also provide a handy tool for network managers to acquire insights of the different network behaviors. This also enables the involvement of network experts in designing and developing machine learning-based systems.
- The utility of network flows as metadata has been shown to increase the learning efficiency and overcome the opaqueness of the data under certain conditions (Haddadi and Zincir-Heywood 2016). This implies that novel data representation and summarization techniques could help to speed up the implementation and evolution of anomaly detection systems.

Cross-References

- ▶ [Big Data for Cybersecurity](#)
- ▶ [Big Data in Computer Network Monitoring](#)
- ▶ [Network Big Data Security Issues](#)

References

- Bhuyan MH, Bhattacharyya DK, Kalita JK (2014) Network anomaly detection: methods, systems and tools. *IEEE Commun Surv Tutorials* 16(1):303–336. <https://doi.org/10.1109/SURV.2013.052213.00046>
- Casas P, Mazel J, Owezarski P (2012) Unsupervised network intrusion detection systems: detecting the unknown without knowledge. *Comput Commun* 35(7):772–783. <https://doi.org/10.1016/j.comcom.2012.01.016>
- Dainotti A, Pescapé A, Ventre G (2007) Worm traffic analysis and characterization. In: 2007 IEEE international conference on communications, pp 1435–1442. <https://doi.org/10.1109/ICC.2007.241>
- Dainotti A, Pescapé A, Ventre G (2009) A cascade architecture for DoS attacks detection based on the wavelet transform. *J Comput Secur* 17(6): 945–968
- Finamore A, Mellia M, Meo M (2011) Mining unclassified traffic using automatic clustering techniques. Springer, Berlin/Heidelberg, pp 150–163. https://doi.org/10.1007/978-3-642-20305-3_13
- García S, Grill M, Stiborek J, Zunino A (2014) An empirical comparison of botnet detection methods. *Comput Secur* 45:100–123. <https://doi.org/10.1016/j.cose.2014.05.011>
- Haddadi F, Zincir-Heywood AN (2016) Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. *IEEE Syst J* 10:1390–1401. <https://doi.org/10.1109/JSYST.2014.2364743>
- Ilgun K, Kemmerer RA, Porras PA (1995) State transition analysis: a rule-based intrusion detection approach. *IEEE Trans Softw Eng* 21(3):181–199. <https://doi.org/10.1109/32.372146>
- Jiang S, Song X, Wang H, Han JJ, Li QH (2006) A clustering-based method for unsupervised intrusion detections. *Pattern Recogn Lett* 27(7):802–810. <https://doi.org/10.1016/j.patrec.2005.11.007>
- Kayacik HG, Zincir-Heywood AN, Heywood MI (2007) A hierarchical SOM-based intrusion detection system. *Eng Appl Artif Intell* 20(4):439–451
- Khanchi S, Heywood MI, Zincir-Heywood AN (2017) Properties of a GP active learning framework for streaming data with class imbalance. In: Proceedings of the genetic and evolutionary computation conference, pp 945–952. <https://doi.org/10.1145/3071178.3071213>
- Kohonen T (2001) Self-organizing maps. Springer series in information sciences, vol 30, 3rd edn. Springer, Berlin/Heidelberg. <https://doi.org/10.1007/978-3-642-56927-2>
- Laney D (2001) 3D data management: controlling data volume, velocity, and variety. Technical report, META Group
- Le DC (2017) An unsupervised learning approach for network and system analysis. Master's thesis, Dalhousie University
- Le DC, Zincir-Heywood AN, Heywood MI (2016) Data analytics on network traffic flows for botnet behaviour detection. In: 2016 IEEE symposium series on computational intelligence (SSCI), pp 1–7. <https://doi.org/10.1109/SSCI.2016.7850078>
- Leung K, Leckie C (2005) Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the twenty-eighth Australasian conference on computer science, vol 38, pp 333–342
- Otey ME, Ghoting A, Parthasarathy S (2006) Fast distributed outlier detection in mixed-attribute data sets. *Data Min Knowl Discov* 12(2–3):203–228. <https://doi.org/10.1007/s10618-005-0014-6>
- Perdisci R, Gu G, Lee W (2006) Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems. In: Sixth international conference on data mining (ICDM'06), pp 488–498. <https://doi.org/10.1109/ICDM.2006.165>

- Rajashekhar D, Zincir-Heywood AN, Heywood MI (2016) Smart phone user behaviour characterization based on autoencoders and self organizing maps. In: ICDM workshop on data mining for cyber security, pp 319–326. <https://doi.org/10.1109/ICDMW.2016.0052>
- Rashid T, Agrafiotis I, Nurse JR (2016) A new take on detecting insider threats: exploring the use of hidden Markov models. In: Proceedings of the 8th ACM CCS international workshop on managing insider security threats, pp 47–56. <https://doi.org/10.1145/2995959.2995964>
- Sequeira K, Zaki M (2002) ADMIT: anomaly-based data mining for intrusions. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, pp 386–395. <https://doi.org/10.1145/775047.775103>
- Shabtai A, Kanonov U, Elovici Y (2010) Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method. *J Syst Softw* 83(8):1524–1537. <https://doi.org/10.1016/j.jss.2010.03.046>
- Sommer R, Paxson V (2010) Outside the closed world: on using machine learning for network intrusion detection. In: 2010 IEEE symposium on security and privacy, pp 305–316. <https://doi.org/10.1109/SP.2010.25>
- Thottan M, Ji C (2003) Anomaly detection in IP networks. *IEEE Trans Signal Process* 51(8):2191–2204
- Veeramachaneni K, Arnaldo I, Korrapati V, Bassias C, Li K (2016) AI2: training a big data machine to defend. In: 2016 IEEE 2nd international conference on big data security on cloud (BigDataSecurity). IEEE, pp 49–54. <https://doi.org/10.1109/BigDataSecurity-HPSC-IDS.2016.79>

Big Data in Smart Cities

Zaheer Khan¹ and Jan Peters-Anders²
¹University of the West of England, Bristol, UK
²Austrian Institute of Technology, Vienna, Austria

Synonyms

Smart cities or future cities; Smart City Big Data or large-scale urban data and/or city data

Definitions

A smart city is a city which invests in Information and Communication Technology (ICT) enhanced governance and participatory processes to develop appropriate public service, transporta-

tion, and energy infrastructures that can ensure sustainable socioeconomic development, healthy environment, enhanced quality-of-life, and intelligent management of environment and natural resources. From a technology perspective, smart cities are focused on capturing, processing, and making effective use of heterogeneous, temporal, and ever increasing large-scale or big urban data.

Overview

City governance is a complex phenomenon and requires new combinations of top-down and bottom-up ICT-enabled governance methodologies to efficiently manage cities. Cities are a major source of urban, socioeconomic, and environmental data, which provide basis for developing and testing Smart City solutions. The notion of Smart Cities has gained a lot of momentum in the past decade. City administrators have realized the potential for ICT to collect and process city data and generate new intelligence concerning the functioning of a city that aims to secure effectiveness and efficiency in city planning, decision-making, and governance. Without urban data and use of ICT, this intelligence either cannot be generated or requires considerable resources, e.g., financial, human, etc.

Recent publications in smart cities domain emphasize better use of urban data by developing innovative domain specific applications which can provide new information or knowledge (Khan et al. 2012, 2014a, 2015; Garzon et al. 2014; Soomro et al. 2016). This new knowledge can be used by city administrations, citizens, and private sector organizations to effectively and efficiently manage their daily activities and/or business processes, as well as the decision-making processes that underpin the governance of the city. The city data is generated from various thematic domains (e.g., transport, land-use, energy, economic, climate, etc.); however, there is little debate on the holistic view of cross-thematic city data which requires Big Data management approaches and high performance computing

paradigms like cloud computing (Khan et al. 2015).

In this entry, we provide a holistic view of Smart City Big Data by reviewing different Smart City projects and applications (UrbanAPI, DECUMANUS, Smarticipate), describing cross-thematic domain Big Data and data management challenges cities face and for which they require appropriate solutions. We also list various datasets available in cities and share lessons learned from the above-mentioned projects.

Smart Cities Big Data: A Holistic View

Smart Cities are tightly coupled with data and information which can be used for knowledge generation to take decisions aiming to resolve societal challenges concerning the quality of life of citizens, economic development, and more generally sustainable city planning. In this section, we try to answer the question: Is Smart City data really Big Data? In this respect, we try to present different facets of Smart Cities data aligned with the main “V” characteristics of the Big Data paradigm, i.e., *variety*, *volume*, and *velocity*. To our knowledge there are several research publications on individual Smart City applications, but a holistic view of the variety of data available in cities is not evident in the current literature.

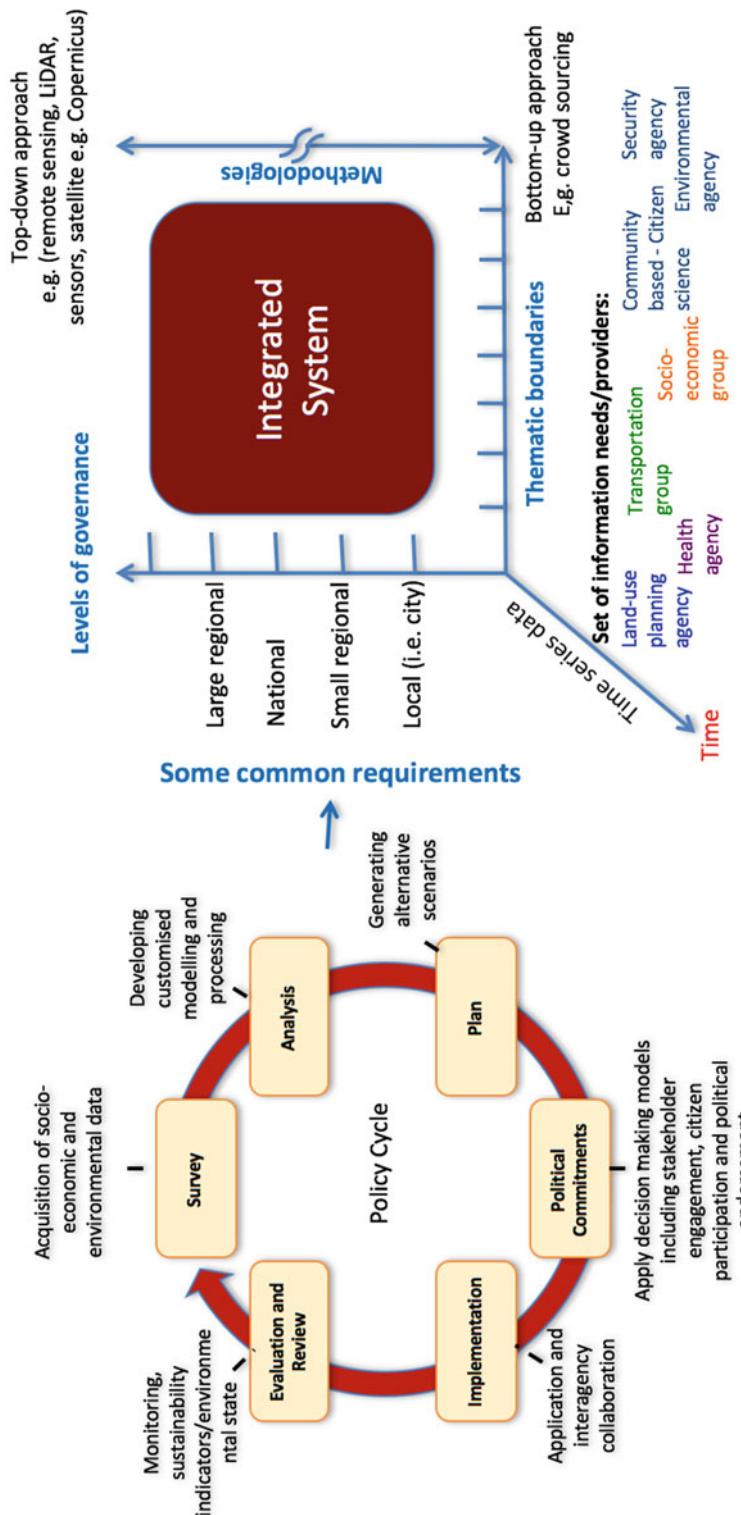
City data includes socioeconomic, built-environment, and environmental data that can be utilized for data analytics to generate useful information and intelligence. Often this data is collected using conventional methods such as daily local business transactions with city administrations, opinion surveys, traffic counters, LiDAR earth observation, sensors for environment monitoring (e.g., noise, air quality, etc.), and well-being and quality of life data from health agencies. More advanced methods like satellite based earth observation imagery (e.g., land monitoring, energy loss, climate change) are also being used to develop the information and intelligence base for analysis across a range of scales from local to city and city-region. Most of the city data has geo-

coordinates which provide location context. This location context helps to perform investigation and data analysis associated with specific areas in cities. This data is stored in different city repositories often managed by different city departments. Some processed data is partly made accessible through open data initiatives (Open Government Data 2017). Often individual elements of open data cannot provide useful information on their own, but when integrated with data from various other data sources (e.g., basemaps, land-use, etc.), it becomes extremely useful to determine the possible effects and impacts of planning interventions that influence the future development of the sustainable Smart City. In addition to the above data sources, new technological inventions such as Internet of Things (IoTs), smart phones (call data records), social media, etc., are also adding to the huge amount of existing urban data.

Figure 1 shows where Smart Cities can make effective use of the city’s Big Data. Cities want to make effective use of data in planning and policy-making processes to make informed decisions. A typical policy-making process generates multiple information requirements, which involves integration of data from different thematic or sectoral domains; addresses the temporal dimension of data; and ensures that data generated at various governance levels, i.e., from city to national and regional levels, is taken into consideration.

This suggests that though application specific data provides necessary information it is not sufficient to provide the holistic overview essential to the monitoring of a city environment by interconnected socioeconomic and environmental characteristics. For instance, land-use and transportation are highly inter-linked, e.g., urban sprawl results in increasing use of the private cars which generates unhealthy particulates affecting air quality in cities, causing health problems for citizens and negative impacts for social/health services. An integrated analysis of the data generated in the urban environment is essential to fully understand the complexities involved in Smart City Big Data governance and management.

In our previous work (Khan et al. 2012), we presented a generic process for integrated



Big Data in Smart Cities, Fig. 1 Integrated view – an example of drivers and impact assessment (Khan et al. 2012)

information system development and identified several technical requirements related to urban and environmental data management including data acquisition, metadata, discovery and metadata catalog, data quality, data retrieval, data storage, schema mapping and transformations, service interoperability, data fusion and synthesis, etc. With the expected growth of IoTs and sensors; extensive use of social media and smart phones; and use of satellite-based very high-resolution imagery for city planning and decision-making, these requirements are still valid for today's Smart City Big Data management.

We present a holistic view of a variety of Smart City data in Fig. 2, classified into two categories: (i) the “Base Data” and (ii) the “Other Thematic/Application specific data.” Typically, cities possess huge amounts of historical data in their databases or electronic files. This data can be related to buildings, base maps, topography, land cover/use, road networks, underground utility/service distribution network, population, green infrastructure, master plan, etc. We refer

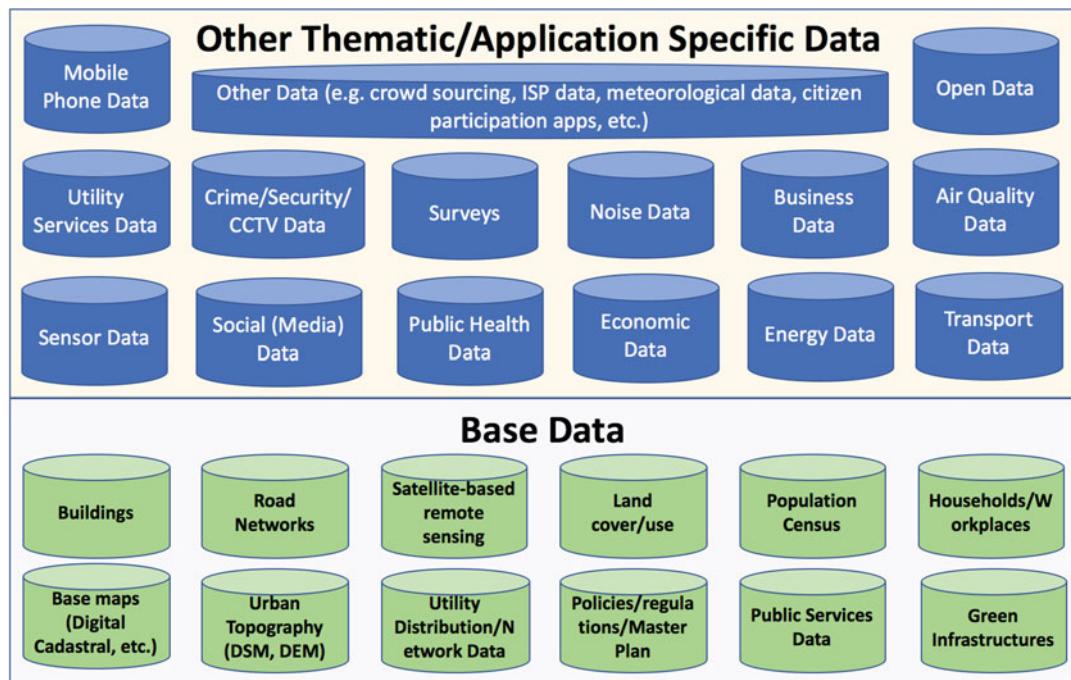
to this data as the base data, which provides the necessary basis for developing Smart City applications. Usually such data is collected by well-tested methods e.g., LiDAR scanning, and hence is trustable (veracity). Such data is usually collected once and updated after many years, e.g., population, road networks, LiDAR based topography. In some cases, a regular update is necessary, e.g., new building constructions. The volume of such data also varies from one city to another. For a medium to large scale city, this data can be from few 100 MBs to 10 s of GBs depending upon preserved historical temporal data (velocity).

The base data may comprise of:

Buildings: 2D/3D building location or structure. This can also be in CityGML format or BIM (Building Information Modeling) compliant data.

Road Networks: include data about major and small road networks, pedestrian or cycle pathways.

Basemaps: provide background maps of a city in different resolutions for orientation purposes.



Big Data in Smart Cities, Fig. 2 Holistic view of smart city big data

Satellite-Based Remote Sensing Data: This can be variety of data collected as high resolution images. This data can be related to light emission, thermal energy loss, and land-monitoring (e.g., land cover changes including green/blue/brown areas).

Land Cover/Use: provides landscape data classified into forests wetlands, impervious surfaces, agriculture, etc. Land cover can be determined by airborne or satellite imagery. In contrast, land-use provides additional information to see land cover changes over time.

Population Census: urban population including density of people (by age and gender) in specific areas.

Households/Workplaces: buildings or places classified as households or workplaces.

Urban Topography: Digital earth images mainly identifying various features. Mostly, it is provided as digital surface model and/or digital elevation model.

Utility Distribution Network Data: Underground sewerage pipes, water distribution pipes, gas/electricity/fiber optic lines, domestic/commercial use of energy, etc.

Policies or Master Plan: Medium to long-term City Master Plan that specifies future urban development.

Green Infrastructure: Tree canopy, parks, front/back gardens, allotments, etc.

Public Services Data: contains information about public services rendered to citizens, e.g., council tax, social housing, social benefits, etc.

The application specific data is more frequently generated (velocity) and belongs to different thematic domains (variety). The temporal dimension of data allows cities to analyze historical changes in the land-use and other applications. For instance, mobile phone data can be in the range of few hundred GBs to TBs per week in unstructured to semistructured format. This data can be used to analyze population distribution and mobility patterns for the daily analysis offering insights concerning the socioeconomic dynamic of the city as a direct input to city planning (Peters-Anders et al. 2017).

As another example, a typical calibrated data reading from a basic sensor can be up to 100

bytes (ceil value) and can be recorded in a CSV or XML file in a semistructured format. As an example, assuming there are 500 sensors deployed in a neighborhood of a city, and each sensor sends data every 10 min, then the total data collected in one hour is: $100 \text{ (approx. bytes)} \times 500 \text{ (sensors)} \times 6 \text{ (for 1 h)} = 300,000$ (about 300 K Bytes). This means total data collected in one day would be: $24 \text{ h} \times 300,000 \text{ bytes} = 7,200,000 \text{ bytes}$ (approximately 7.2 M Bytes), and the total data collected in one year $7.2 \text{ MB (data per day)} \times 365 \text{ (days / year)} = 2.628 \text{ G Bytes}$. It is clear, as there are a variety of other sensors (e.g., infrastructure monitoring sensors, air quality – PM10, NO₂, etc.) in the city environment collecting different thematic data on daily basis, that the daily data collection may reach 100 s of G Bytes to 10 s of T Bytes.

The above example considers only one source of data and there are several other sources like CCTV, energy consumption by households, transport data, social media, etc., which results in the generation of a few 100 s of GBs to 10 s of TBs data on daily basis. The amount of this data will significantly increase with deployment of 1000 s of new sensors, use of smart devices, and billions of Internet of Things (IoTs) in the next few years.

The thematic/application specific data may comprise of:

Mobile Phone Data: call data records (CDR) are generated when a mobile phone user performs any activity on a smart phone, e.g., calling, texting, etc. This data is normally collected by service providers and can be unstructured or semistructured data.

Utility Services Data: energy consumption, fuel (domestic methane gas, vehicle fuel) usage, water consumption, etc. This can be semistructured data.

Sensor Data: variety of sensors including air quality, traffic counters, etc. This can be unstructured data.

Social Media Data: from social media, Twitter, Facebook, etc. This is unstructured data.

CCTV Data: from CCTV footage, transport live feed, etc. This is unstructured data.

Security Data: crime/incidents at different areas represented as semistructured or structured data.

Public Health Data: health records, health surveys, etc. This can be semistructured or structured data.

Economic Data: public financial records as well as overall financial data of city. This can be semistructured to structured data.

Survey Data: survey carried out in different districts about quality of life, public services, etc. This can be unstructured as well as semistructured data.

Business Data: data about local enterprises/businesses, etc. This can be unstructured (e.g., business website) or semistructured data.

Energy Data: data from smart meters, energy grid (demand/supply), etc. This can be semistructured or structured data.

Transport Data: data about public/private transport, traffic counters, traffic cameras, fuel consumption for mobility, etc. This can be unstructured or semistructured data.

Air Quality Data: particulate concentration in the air – PM10, CO, etc. This can be unstructured or semistructured data.

Open Government Data: Data published by the cities for open use by citizens and local businesses. A variety of formats and datasets may be provided and can be unstructured, semistructured, or structured.

In general, the city datasets can be available as unstructured, semistructured, and structured data, and in a variety of formats including PDF, XLS, XML, CSV, SHP, DRG, GeoTIFF, JPEG2000, X3D, GeoJSON, CityGML, etc., or be published online via, e.g., OGC (Open Geospatial Consortium) web-services, like WMS, WMTS, WFS, etc.

Structured data is normally held in a relational data model and stored in databases of different departments or agencies of the city administration. Examples of such data can be land cadastre, population census, public health, energy consumption, vector data, e.g., road networks, etc.

The semistructured data is often stored in file format like, e.g., CSV, XML, JSON, or NoSQL databases. Often application specific high fre-

quency data is stored in the above formats. Such data can be queried by writing specific scripts to generate the required information.

The unstructured data can be social media data (e.g., Twitter), citizens comments on planning applications, audio/video, live traffic feeds (e.g., CCTV), satellite/raster imagery, city master plan or policy documents (e.g., PDF), radar or sonar data (e.g., vehicular, meteorological data), surveys, mobile data, etc. Such data can be in text, audio/video, or image format and hence it is challenging to query such data.

The above discussion highlights the following important aspects:

- (i) Cities generate large amount of data on daily basis which possess all “V” Big Data characteristics
- (ii) Location based (i.e., geo-coordinate related) data is a key element in Smart City Big Data to generate contextual information
- (iii) City data may come in unstructured, semistructured, or structured formats and requires innovative Big Data processing approaches

Applications and Lessons Learned

Here we present key Smart Cities Big Data applications and associated data challenges which need investigation for the development of Smart City information systems.

Applications

The Framework Programme (FP) Seven UrbanAPI project (Khan et al. 2014b) used both base data and thematic/application specific data, which provides a suitable example of the use of unstructured, semistructured, and structured datasets. For example, Peters-Anders et al. (2017) used mobile data (CDR) to gain new insights in the urban environment. It was observed that the mobile data collected by telecom operators may range into few 100 s of GBytes to TBytes on weekly basis. Mostly data is received in customized binary format or in proprietary file formats and millions of records are stored in

sequential (unstructured) order. Processing of such data requires clear understanding of the file formats to extract the required information. It was also observed that the quality and accuracy of the time series data needs to be checked before processing and drawing conclusions on the outcomes. Peters-Anders et al. (2017) explained various base datasets used with the mobile data to generate an interactive visual interface depicting daily population distribution and mobility patterns. The authors highlighted data quality (e.g., difference in temporal and spatial resolution; lack of fine granularity in mobile data; privacy issues), processing issues (e.g., lack of a common data model), and suggested advanced techniques, e.g., a Map-Reduce model or cloud computing, that can be applied to generate required outputs.

In contrast, The FP7 DECUMANUS project (Garzon et al. 2014) provides a suitable example of Big Data application where satellite based very high resolution images are collected, often in 100 s of M Bytes to G Bytes, for the development of Smart City services. For some services, e.g., climate service, data could reach to 100 s of G Bytes to T Bytes. One of the positive aspects of the FP7 DECUMANUS project was that it demonstrated practical use of satellite based earth observation data in city planning (Ludlow et al. 2017). The higher resolution imagery; the capacity to capture more updated images on frequent basis; lower cost; and the potential to complement any gaps by integrating it with in-situ data were the key success factors. However, the data comes in unstructured form and complex algorithms have to be used for digital processing and integration with in-situ data (e.g., base or thematic data). Different DECUMANUS services demonstrated this complexity, e.g., climate change downscaling processing (San José et al. 2015a), health impact (San José et al. 2015b); urban green planning (Marconcini and Metz 2016); and energy efficiency. For instance, San José et al. (2015a) describes the experiments where very high-resolution earth observation images, i.e., covering about 23×20 Kilo meters (km) at European Latitude, were dynamically down-scaled to about 200 and 50 m resolution scales,

which is more suitable to visualize and analyze data at city scale. This downscaling was performed by using the Barcelona Super Computing's high performance computing infrastructure (128 nodes/blades; each blade with 8 GB RAM and two 2.3 GHz processors; Global Parallel File System; Linux OS and Myrinet high bandwidth network to connect nodes) and consumed about 881,664 CPU hours.

As compared to the above applications, the Horizon 2020 smarticipate project makes use of base and thematic datasets to develop citizens participation applications and generate new datasets for public services (Khan et al. 2017a). The smarticipate platform enables cities to create topics of interest, i.e., a specific planning proposal in a neighborhood, and share it with citizens to seek their opinions, suggestions, and alternative proposals with the objective to co-create and/or co-design local neighborhood plans. The topic creation requires datasets from base data (e.g., base maps, layers of information, e.g., road networks, street furniture, etc.) and application/thematic data, e.g., air quality data, soil data, noise, utility infrastructure, etc. For example, Hamburg's tree planting use case requires base maps with street networks, soil data, utility infrastructure, and existing tree data. All this data can be in 2D as well as in 3D to provide visual representation of the area. This means topic creation can make use of structured and semistructured data. Once citizens are invited to participate for a specific topic, they can provide Web2.0 based comments by engaging in a dialogue with other citizens (unstructured data), showing likeness/dis-likeness or creating alternative proposals by adding more data on the platform.

It is expected that a typical topic creation may add 10 s of MBytes to 100 s of MBytes (3D scene) and may seek citizens' participation for a full week (7 days). Let us assume 50 MBytes is used for tree planting topic creation. Now imagine on average each user adds about 1000 bytes of comments to one proposal, and if we assume that about 1000 citizens provide their comments, the total amount of unstructured data generated will be $1000 \times 1000 = 1$ MBytes approximately. This

means one instance of a proposal can generate about 1MBytes of unstructured data, so total data for one instance will become 50 MBytes (topic) + 1 MBytes (comments) = 51 MBytes. If half of the sample group create alternative proposals (with same base/thematic or new data) and attract new comments on the alternative proposals, then the data generated will be: 51 MBytes (data from main topic/proposal) \times 500 (alternative proposals) = 25.5 GBytes. This means, typically, about 25 GBytes data per week is expected. Imagine if there are 10 different topics initiated by the city council in different neighborhoods, then it is expected to generate about 250 GBytes of data per week. This amount may further increase if the topic creation is more complex such as the Forte Trionfale refurbishment scenario (i.e., with buildings and other surrounding infrastructure) for the city of Rome, where over 100 MBytes may be used for topic creation. Over time the amount of data generated by the smarticipate platform will increase significantly and to be effective must utilize Big Data management and processing capabilities by using Edge, Hadoop, Spark and Storm, etc., based approaches.

Challenges as Lessons Learned

Structuring of Data: The structuring of data varies from one city to another, i.e., one city may have a specific dataset in unstructured format, whereas another city may have designed it in well-structured format. For example, most cities have their city master plan in a static format, e.g., image or document. However, cities with more advanced data collection and management resources (e.g., Hamburg in the smarticipate project) have their city master plan defined in an Object Oriented Data Model that provides more flexibility to query the plan and perform monitoring and validation of land-use development.

The spatial data is often represented in a specific spatial reference system. Cities in different countries use different spatial reference systems which may require Smart City applications to accommodate different reference systems for harmonizing and processing the data and providing the information in a uniform representation, e.g., to perform comparative analysis.

Cities collect new data on regular basis and provide temporal time series data that can be used to detect changes, e.g., land-use, built environment or atmosphere, etc. For example, land monitoring data (e.g., LiDAR) can be collected bi-yearly due to data collection costs; likewise, satellite based remote sensing data can be collected more frequently, e.g., bi-monthly; in contrast, mobile data, e.g., Call-Data-Records contain data instances based on any mobile phone activity and can be in seconds, minutes, or hours.

All the above applications indicate that there is no standard approach defined and adopted by the cities to manage the temporal and spatial dimension of city data, which makes it difficult to develop generic processing scripts or applications which can be reused by many cities. This highlights the need to use standard data representation or exchange formats, like CityGML, which can tackle variety of data format heterogeneity by using its so-called Application Domain Extension (ADE) to represent energy, utilities, etc.

Quality of Data: The base data is often stored in a non-normalized schema; have different data formats; and is managed/owned by different departments or agencies. In addition, often there is incomplete or no meta-data available and not all the data comes with geo-coordinates. Data may have different spatial resolution scales and often data is not available in the required spatial scale, e.g., up to few centimeters. All these aspects compromise the quality of the data and require rigorous data harmonization and compatibility efforts and preprocessing to prepare the data for possible use in different thematic applications.

Veracity of Data: The level of accuracy of the base and thematic data may vary from one application to another. Most base data are collected by using reliable data collection methods, e.g., LiDAR, official population census, etc. However, not all the thematic applications relying on the data from citizen science or public participation, down-scaled satellite imagery, etc., provide highly accurate data, and this suggests the need to validate the data source to make it trustworthy, reliable, and usable in other applications or in decision-making processes. For instance, W3C PROV model (2013) is used in citizen participa-

tion applications to validate the source of data and establish trust in it (IES Cities 2013).

Data Access and Availability: A lot of data is managed in different departmental silos. This data is often without standard accessible interfaces or APIs (e.g., OGC's WFS interface), which makes it difficult to discover/search or access from the external world. As a result, cities might have duplicate and often nonmatching data in different departments or agencies.

Data Security and Privacy: Security and privacy related concerns right up the agenda in response to security and privacy related legislations, e.g., the Data Protection Act 1996 or the General Data Protection Regulations (GDPR). While suitable harmonizing and integration and processing approaches are needed for city data, this must not accidentally violate individuals' privacy. This becomes really challenging when multiple data sources are combined for Smart City Big Data processing. Also, Smart City services should provide end-to-end security and privacy by adopting novel security frameworks, e.g., SSServProv Framework (Khan et al. 2017b) or blockchain based authentication and self-verification techniques, e.g., VeidBlock (Abbasi and Khan 2017).

Data Processing and Data Transfer: Many applications are developed by using OGC compliant services. Use of Apache Hadoop and Spark and Storm based platforms permit the processing of historical as well as real-time data (Khan et al. 2014a, 2015; Soomro et al. 2016). Other libraries and tools for geo-processing (e.g., Java based GeoServer on the back-end and JavaScript based mapping libraries like, e.g., OpenLayers, Leaflet, etc., in the front-end) can be utilized to process and present city data to different stakeholders (Khan et al. 2017a). In some cases, micro-services or a typical service oriented environment can be used to design service workflows; data can be accessed and processed by using OGC compliant WFS and WPS interfaces. In most cases, data is fetched from remote repositories, transferred, and stored in central repositories (or on the cloud) to process the data (Khan et al. 2012). In such situations, a transfer of huge amounts of data

takes place which is not sustainable, and hence requires alternative distributed computing models such as fog computing (Perera et al. 2017) and/or edge computing (Ai et al. 2017).

Summary and Future Directions for Research

This entry provides a thorough insight into Smart City Big Data covering "V" characteristics and main categories: basic and thematic data. From the Smart City Big Data perspective, we presented three applications from FP7 UrbanAPI, FP7 DECUMANUS, and H2020 smarticipate projects. We also highlighted various data related challenges cities are facing today including city data possess harmonization, integration, and processing, which require further research to apply distributed or edge computing models.

The applications and challenges provide pointers to future research directions including: (i) designing common data models for data harmonization and application development; (ii) defining and using standard approaches for data representation or exchange, e.g., CityGML; (iii) inventing new edge based models so that processing can be performed close to the data source and unnecessary data transfer can be limited; (iv) providing necessary security and privacy measures to avoid violation of privacy laws; (v) assessing and improving data quality and provenance through new techniques; (vi) introducing intuitive visualization of data for better awareness raising; and (vii) innovative data analytics approaches to generate new information and knowledge to support decision-making in city governance.

Cross-References

- ▶ [Big Data Analysis for Smart City Applications](#)
- ▶ [Big Data Analysis and IOT](#)
- ▶ [Linked Geospatial Data](#)
- ▶ [Query Processing: Computational Geometry](#)
- ▶ [Spatio-Textual Data](#)
- ▶ [Spatio-Social Data](#)

- **Streaming Big Spatial Data**
- **Spatiotemporal Data: Trajectories**

Acknowledgments Authors acknowledge that the work presented in this chapter was funded by the University of the West of England, Bristol, UK and three European Commission projects: FP7 UrbanAPI (Grant# 288577), FP7 DECUMANUS (Grant# 607183) and H2020 Smar-ticipate (Grant# 693729). We also thank David Ludlow from the University of the West of England, Bristol, UK for reviewing and providing constructive feedback on this chapter.

References

- Abbasi AG, Khan Z (2017) VeidBlock: verifiable identity using blockchain and ledger in a software defined network. In: SCCTSA2017 co-located with 10th utility and cloud computing conference, Austin, 5–8 Dec 2017
- Ai Y, Peng M, Zhang K (2017) Edge cloud computing technologies for Internet of things: a primer. *Digital Commun Netw.* <https://doi.org/10.1016/j.dcan.2017.07.001>
- Garzon A, Palacios M, Pecci J, Khan Z, Ludlow D (2014) Using space-based downstream services for urban management in smart cities. In: Proceedings of the 7th IEEE/ACM utility and cloud computing, 8–11 Dec 2014. IEEE, London, pp 818–823
- IES Cities (2013) Internet-enabled services for the cities across Europe, European Commission, Grant Agreement 325097
- Khan Z, Ludlow D, McClatchey R, Anjum A (2012) An architecture for integrated intelligence in urban management using cloud computing. *J Cloud Comput Adv Syst Appl* 1(1):1–14
- Khan Z, Liaquat Kiani S, Soomro K (2014a) A framework for cloud-based context-aware information services for citizens in smart cities. *J Cloud Comput Adv Syst Appl* 3:14
- Khan Z, Ludlow D, Loibl W, Soomro K (2014b) ICT enabled participatory urban planning and policy development: the UrbanAPI project. *Transform Gov People Process Policy* 8(2):205–229
- Khan Z, Anjum A, Soomro K, Muhammad T (2015) Towards cloud based big data analytics for smart future cities. *J Cloud Comput Adv Syst Appl* 4:2
- Khan Z, Dambruch J, Peters-Anders J, Sackl A, Strasser A, Frohlich P, Templer S, Soomro K (2017a) Developing knowledge-based citizen participation platform to support Smart City decision making: the smarticipate case study. *Information* 8(2):47
- Khan Z, Pervaiz Z, Abbasi AG (2017b) Towards a secure service provisioning framework in a Smart City environment. *Futur Gener Comput Syst* 77:112–135
- Ludlow D, Khan Z, Soomro K, Marconcini M, Metz A, Jose R, Perez J, Malcorps P, Lemper M (2017) From top-down land use planning intelligence to bottom-up stakeholder engagement for smart cities – a case study: DECUMANUS service products. *Int J Serv Technol Manag* 23(5/6):465–493
- Marconcini M, Metz A (2016) A suite of novel EO-based products in support of urban green planning. In: Conference on urban planning and regional development in the information society, 22–24 June 2016
- Open Government Data (2017) Open knowledge foundation [online], URL-Access: <https://okfn.org/about/our-impact/opengovernment/>. Last-accessed 16 Oct 2017
- Perera C, Qin Y, Estrella J, Reiff-Marganiec S, Vsilakos A (2017) Fog computing for sustainable smart cities: a survey. *ACM Comput Surv* 50(3):1–43. 32
- Peters-Anders J, Khan Z, Loibl W, Augustin H, Breinbauer A (2017) Dynamic, interactive and visual analysis of population distribution and mobility dynamics in an urban environment using mobility explorer framework. *Information* 8(2):56
- San José R, Pérez J, Pecci J, Garzón A, Palacio M (2015a) Urban climate impact analysis using WRF-chem and diagnostic downscaling techniques: Antwerp, Helsinki, London, Madrid and Milan case studies. In: Proceedings of the VII atmospheric science symposium, Istanbul, vol 27, pp 223–247, Apr 2015
- San José R, Pérez J, Pérez L, González R, Pecci J, Garzón A, Palacios M (2015b) Regional and urban downscaling of global climate scenarios for health impact assessments. *Meteorol Clim Air Qual Appl* 27:223–247
- Soomro K, Khan Z, Hasham K (2016) Towards provisioning of real-time Smart City services using clouds. In: 9th international conference on utility and cloud computing, Shanghai, 06–09 Dec 2016
- W3C PROV model (2013) W3C PROV model primer, W3C working group note 30 Apr 2013. Online, URL at: <https://www.w3.org/TR/prov-primer/>. Last-Accessed 12 Oct 2017

B

Big Data in Social Networks

Antonio Picariello

Università di Napoli Federico II – DIETI,
Napoli, Italy

Synonyms

Big data analytics; Social networking

Definitions

Online social networking is a novel paradigm within the big data analytics framework, where massive amounts of information about heterogeneous social facts and interactions are stored, at a very high variability rate. By considering important problems such as influence diffusion and maximization and community detection, we show the importance of this research field in big data analysis.

Overview

We describe the main research activities in big data social networking, focusing on influence diffusion and maximization and community detection performed on on-line social networks.

Social Networks as Big Data

Nowadays, we can say undoubtedly that social networking is one of the main applications of big data techniques and tools. *Online social networks* (OSN) usually produce a collection of very huge data sets (*volume*) with a great diversity of types (*variety*) and sometimes with a high frequency of variability (*velocity*). In addition, how to manage the *veracity* of such kind of data is still a challenge, and the use of social data may have a great economic and competitive impact (*value*). Consequently, in social networking all the 4 + 1 V that characterize big data are really addressed. The rapid development of the application of big data techniques to social networks (sometimes also called *social networking big data*) is bringing revolutionary changes to our daily lives, producing a novel global business. Just to mention an example about one of the most popular OSNs, in terms of volume, in June 2017, Facebook has 2 billion users per month, with hundred billion friendship edges; Facebook adds 500,000 new users every day and 6 new profiles every second; 72% of all US online adults visit Facebook at least once a month. Worldwide, 38.6% of the online population use Facebook. In terms of ve-

locity, Facebook, respectively, generates about 500 Tbytes of data *daily*: Twitter a little less. And in addition these data (text, images, video, audio) also need to be processed at a high speed. In this framework, social network management and analysis evolve into a big data problem when IT specialists try to predict behavior (predictive analytics) or to suggest decisions for getting advantages of the analysis (prescriptive analysis). Clearly, the novel paradigm of social networking and big data should provide approaches for efficiently adopting effectively big data analytic algorithms. In particular, due to the complexity and diversity of OSNs and the produced data, there are two important aspects that need to be deeply investigated: (a) how to conduct social network analysis considering big data and (b) how to use big data analytics technique with respect to privacy and security issues.

In the next sections, we will first describe into details online social networks and related models, and we will focus on social network analysis describing influence diffusion and maximization and community detection techniques.

Online Social Networks

Currently, most social networks connect people or groups who expose *similar* interests or features. However, such networks could surely connect in the next future other entities, such as software, web services, and data repositories (Tan et al. 2013).

Let us now consider a social network as a group of people connected through social relationships: an online social network service is an online platform where people create social relationships with other people sharing similar interests. Usually the interactions between them are performed through media contents: in this case, we sometimes refer to *social media networks*, usually subdivided into the following categories:

- *Online social networks and multimedia social networks*: these networks are based on user-

generated contents. Hence, it is possible to exploit the relationships established among users, users and contents for different purposes, such as (i) identify the most appropriate content for specific users, (ii) ranking of social network users with respect to a given item, and (iii) identify a subset of user to maximize the influence of specific contents and so on. The following – famous – social networks lie into these categories: Twitter, Facebook, Instagram, Google+, Last.fm, and Flickr.

- *Location-based social networks*: These networks introduce location as a new object. Some examples are Foursquare, Yelp, and TripAdvisor.
- *Social movie rating businesses*: In this category, we can identify the following social networks: Flixster, IMDb, and MovieLens.

Social networking has its beginnings in the work of social scientists in the context of human social networks, mathematicians and physicists in the context of complex network theory, and, most recently, computer scientists in the investigation of information social networks. New challenges and research topics are thus arising, especially concerning marketing applications like recommendation, influence maximization, influence diffusion, viral marketing, word of mouth, etc. and also problems related to security like Lurker detection and malicious user detection (Amato et al. 2016a).

Social Networks Models

Graphs are the most suitable data structure that can reflect and represent social networks: a graph can exactly depict the relationships among different users. How to model an OSN with a graph is dependent on the different kinds of objects forming the network.

Generally speaking, a social network is a directed graph $G = (V, E)$, V being the set of vertices of the network – e.g., users, multimedia objects, tags, and so on – and E the set of edges among vertices, e.g., the social ties.

The different entities that could be considered in an OSN typically are *users*, the set of users

that are the subject of the analysis, every user corresponding to a node of the graph (call this set U); *social relationships*, the set of relationships among two users; we can refer to it as the set of edges E ; *multimedia object*, the media contents generated by users and through which users can interact among them; and *tags*, the keywords associated with a particular user-generated content.

In the last few years, the scientific community focused a particular interest into *multimedia social networks*, i.e., the OSN added with multimedia user-generated content. In this case, we can detect different relationships: *user to user*, this is the most common relationship, where two users are connected to each other (e.g., friendships, memberships, following, and so on); *similarity*, this kind of relationships exists between two multimedia objects and refers to the similarity of these contents; it is based on the analysis of high-level and low-level features; and *user to multimedia*, this represents the actions that a particular user can make on a particular content.

In order to handle in a compact way the different kinds of data contained in a OSN, a powerful data structure that is emerging is the hypergraph (Amato et al. 2016b). A hypergraph is the generalization of the concept of graph, in which edges are not referred only to two vertices, but to an undefined number of vertices. Formally, a hypergraph is defined as $H = (X, E)$, X being a set of vertices and E a set of non-empty subsets of X , called hyperedges or edges. Therefore, E is a subset of $2^X \setminus \{\emptyset\}$, where 2^X is, as usually, the power set of X .

Social Network Analysis

When applied on an OSN, the big data analytics techniques may provide what is sometimes called “the wisdom of the crowds” (i.e., the collective opinion of a group of individuals rather than that of a single expert), thus revealing sentiments, detecting typical patterns, providing recommendations, and influencing people or groups of people. Social networks analysis, or SNA, is one of the key points of the modern sociology, especially for the study of society and social relationship between people. Nowadays, the growing popularity of OSNs and the data generated by them lay

the foundations for analyzing several sociological phenomena that are used in many fields. SNA involves several techniques like machine learning, text analysis, and media analysis. Analyzing such huge amount of data allows to understand the social tie between nodes. The chance to analyze this huge amount of data is really important for many kinds of application in very different fields. A summary of these applications are *recommendation systems*, which are the products to suggest to users, based on *collaborative filtering* in which the products to suggest are based on the interests of similar users (Amato et al. 2017b); *influence analysis*, the analysis of the most influential users in the network (Amato et al. 2017a) *community detection*, the detection of different communities of similar users in a network; and *advertising systems*, which ads should be promoted to users.

Briefly, SNA may be broadly divided into two categories: (a) *linkage-based and structural analysis*, the analysis of links among nodes of network to understand social relationships, and *content-based analysis*, the analysis is based on contents generated in the OSN.

Influence Diffusion

Influence diffusion analysis describes how information propagates across the users in a network. The main goal of this research field is to create a diffusion model that describes how information could be “diffused.” Different models have been proposed in the literature, which can be classified into two categories: *progressive* and *nonprogressive* models. The nonprogressive models make the assumption that nodes’ states can change indefinitely, while in the progressive ones, when a node is active, it cannot switch its status. For both the described categories, it’s possible to classify the proposed models into *stochastic*-, *epidemiologic*-, and *voting*-based models.

The stochastic diffusion models are based on stochastic processes; in this case, considering an OSN modeled as a graph, a node can have only two possible states: *active* and *inactive*, depending on if the node has been influenced by the new information or not. Stochastic diffusion models have two properties related to the *influence spread function*: (a) *monotonicity*, meaning

that if we add more nodes to a *seed set*, we cannot reduce the size of the final set, and (b) *sub-modularity*, meaning that a function has the property of the *diminishing returns*. Note that sub-modularity tells us that the obtained gain adding a node v to the total set T is always lower than – equal – adding that node v to a subset S . In mathematical terms:

$$f(S \cup v) - f(S) \geq f(T \cup v) - f(T) \quad (1)$$

The most used stochastic diffusion models are the *independent cascade model* (IC) and the *linear threshold model* (LT).

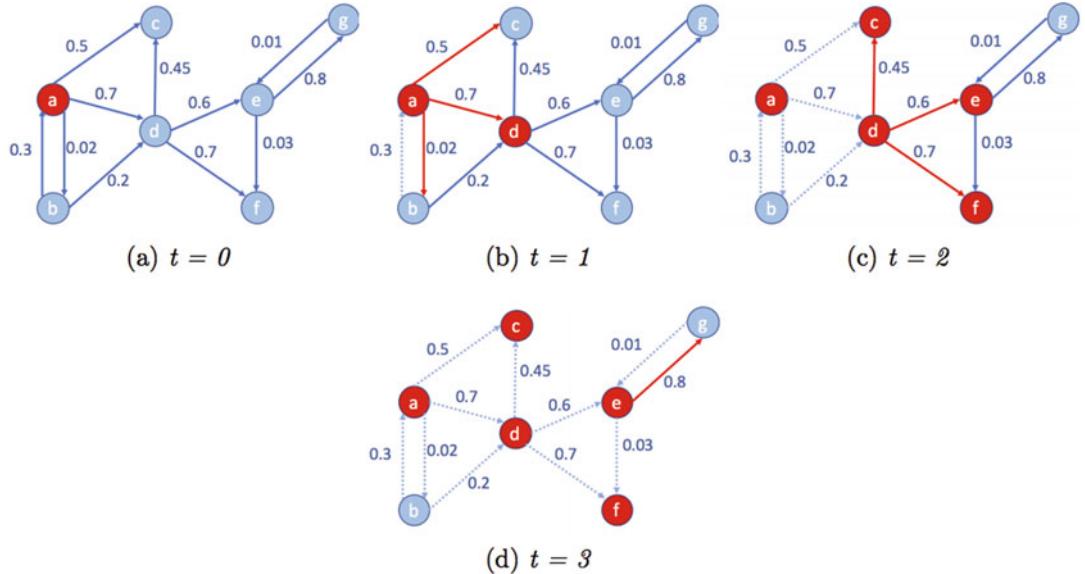
The IC model is described by Kempe et al. 2003a based on the and marketing research Goldenberg et al. 2001. This model is also related to epidemic models (Keeling and Eames 2005).

In this approach, every edge $e_{u,v} \in E$ has a probability $p_{u,v} \in [0, 1]$, and the model works as following:

1. at initial step $t = 0$, an initial seed set S_0 is activated, while the remaining nodes are inactive;
2. each active node u executes an *activation attempt* by performing a *Bernoulli* trial with the probability $p_{u,v}$ to active its neighbors v_i ;
3. if a node v has not been activated, the node u has no others chances to achieve this. If the node v is activated we can add this node to the active sets S_t in the following step $t + 1$. When a node is activated, it can’t turn inactive again;
4. the steps 1–3 are repeated until no new nodes become activated, at time t_n .

This kind of diffusion is very useful to explain the diffusion behavior of information or viruses, like in *contagion*.

Figure 1 shows an example of this diffusion process. At time $t = 0$ only the node a is active Fig. 1a, and it represents the seed set. At step $t = 1$, node a attempts to activate nodes b , c , and d Fig. 1b, and only d is activated; in this case a fails to activate b and c . At this point, the edges used or that have a destination already active, are left out. At step $t = 2$ only d tries to activate its neighbors c , e , and f Fig. 1c. In the



Big Data in Social Networks, Fig. 1 Independent cascade algorithm. (a) $t = 0$. (b) $t = 1$. (c) $t = 2$. (d) $t = 3$

last step, $t = 3$ node e tries to affect g but fails, and the diffusion stops because there are not any outgoing edges to inactive nodes. Fig. 1d. The main problem of this model is that we assume that in each step, a node can be activated only by one of its neighbors, and so the influence of a node depends only on one source.

To overcome the constraints that the influence of a node depends only on one source, scientists considered a novel approach where a node u is activated when it receives influence by more than one neighbor, also known as the LT model, proposed by Kempe et al. 2003b. In this model, every node u has a weight $w(v) \in [0, 1]$ that indicates the threshold to reach to influence the node v . This means that to activate the node v , the sum of the probability $p_{u,v}$ of its active neighbors has to be more than w_v . This corresponds to say that a single node activation depends on multiple sources. Initially, only an initial seed set S_0 is active. The steps of the algorithm are the following:

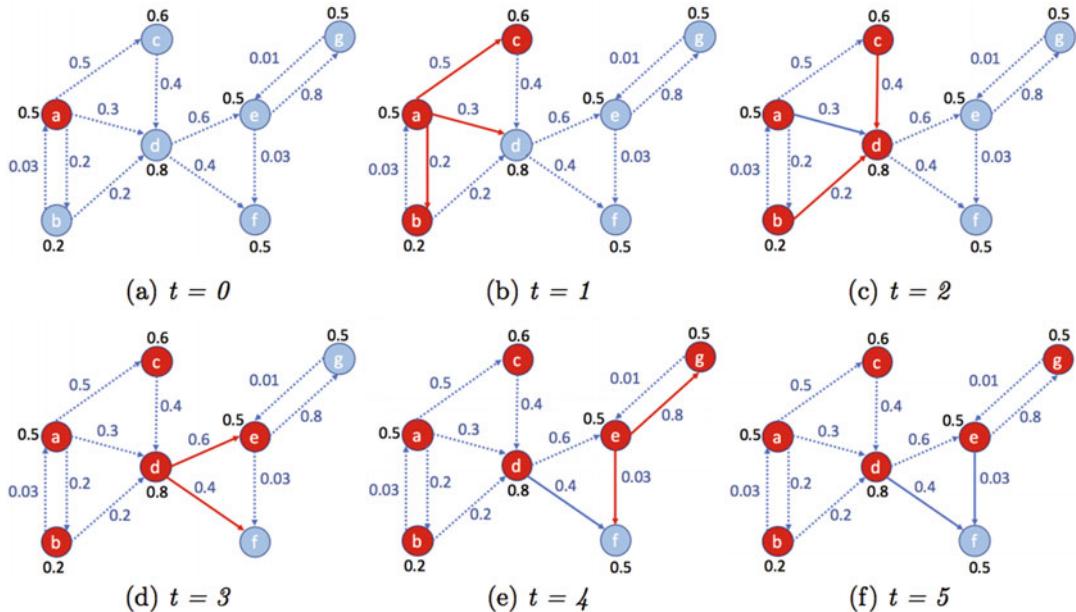
1. at each step t_i every node $u \in S_{t_i}$ attempts to activate its inactive neighbors v_i with a probability $p_{u,v}$, where S_{t_i} indicates the active set at time t_i . The node v becomes active if

the sum of the probabilities is greater than its threshold. If we indicate $N(v)$ as the neighbors nodes of v , it becomes active if:

$$\sum_{v \in N(v) \cap S_{t_i}} p_{u,v} \geq w_v \quad (2)$$

2. if the node v became active, it belongs to the S_{t_i+1} , and it can attempt to influence its neighbors. If it doesn't become active, other new active nodes can contribute to activate it in further steps.

Figure 2 shows an example of LT diffusion process. As in the IC model, when a node becomes active, we mark it with red color; indeed near each node v is shown the threshold w_v . At step $t = 0$, only the node a is active Fig. 2a, it begins to influence its neighbors, but it succeeds only with c and b because their threshold is lower than $p_{a,b}$ and $p_{a,c}$ Fig. 2b. The $p_{a,d}$ is not sufficient to activate d , but it goes on to influence also at step $t = 2$ where, through the influence of c and b , the threshold is reached and node d is activated. The process goes on until there are not nodes become active Fig. 2c.



Big Data in Social Networks, Fig. 2 Linear threshold model. (a) $t = 0$. (b) $t = 1$. (c) $t = 2$. (d) $t = 3$. (e) $t = 4$. (f) $t = 5$

Influence Maximization

The *influence maximization* problem in an OSN requires to find the k nodes having maximum influence. Initially, let us consider a seed set S_0 , formed by all the nodes that can activate a huge number of nodes in the network. Kempe et al. 2003b proved the NP-hardness of this problem. The general approach to this problem is to define a *greedy* algorithm that can get an approximation of the optimal solution.

Kempe demonstrated the following fundamental theorem.

Theorem 1 For a nonnegative, monotone submodular function f , let S be a set of size k obtained by selecting elements one at a time, choosing an element that provides the largest marginal increase in the function value. Let S^* be a set that maximizes the value of f over all k -element sets. Then $f(S) \geq (1 - 1/e) \cdot f(S^*)$; in other words, S provides a $(1 - 1/e)$ -approximation.

There are several approaches to this problem, as in the following categories:

- *Stochastic*: most known approaches are Monte-Carlo based (Kempe et al. 2003a), TIM (Tang et al. 2014, 2015).
- *Biologically inspired*: bio-inspired approaches, mainly based on bee waggle dance model and ant colony model (Sankar et al. 2016).
- *Game theory based*: based on game theory; one of the most important is multi-armed bandit theory (Vaswani et al. 2015);
- *Genetic algorithms*: based on genetic algorithm and so more efficient than greedy approach (Bucur and Iacca 2016);

Community Detection

The modern science of networks has brought significant advances to our understanding of complex systems. One of the most relevant features of graphs representing real systems is community structure, or clustering, i.e., the organization of vertices in clusters, with many edges joining

vertices of the same cluster and comparatively few edges joining vertices of different clusters. Such clusters, or communities, can be considered as fairly independent compartments of a graph. Detecting communities is of great importance in sociology, biology, and computer science, where usually systems are often represented as graphs. This problem is very hard and not yet satisfactorily solved, despite the huge effort of a large interdisciplinary community of scientists working on it over the past few years. In graph theory, community detection is a fundamental task to analyze and understand the structure of complex network. As reported in Wang et al. (2015), “revealing the latent community structure, which is crucial to understanding the features of networks, is an important problem in network and graph analysis.”

During the last decade, many approaches have been proposed to solve this challenging problem in diverse ways. The term “community” has been extensively used in the literature in different contexts and with different connotations.

A first definition is provided by Gulbahce and Lehmann 2008 that define a community as “a densely connected subset of nodes that is only sparsely linked to the remaining network.” Porter et al. (2009) provide a definition based on the fields of sociology and anthropology, in which a community is a “cohesive group of nodes that are connected more densely to each other than to the nodes in other communities.” Moreover, Fortunato 2010 defines communities as “groups of vertices that probably share common properties and/or play similar roles within the graph.” The most commonly used definition is that of Yang et al. (2010): “a community as a group of network nodes, within which the links connecting nodes are dense but between which they are sparse.” Papadopoulos et al. (2012) ultimately define communities as “groups of vertices that are more densely connected to each other than to the rest of the network.”

However, the definition of a community is not clear because it is complex to define the concept of “good” community. Several possible

communities can be computed in a social network $G(V, E)$ whose enumeration is a NP-complete problem. The aim of several community detection algorithms is to optimize a goodness metric that represents the quality of the communities detected from the network.

It is possible to define the two main types of community. A *strong community* is a subgraph in which each vertex has more probability to be connected with the vertices of the same community compared to a node in a different community. A *weak community* is a subgraph in which the average edge probability between each vertex in the same community exceeds the average edge probability between the vertex with the vertices of any other group.

As shown in Chakraborty et al. (2017), usually community detection analysis is composed of two phases: detection of meaningful community structure from a network and, second, evaluation of the appropriateness of the detected community structure. In particular, the second phase raises the importance to define several metrics that are the fundamentals of different community detection definitions. Moreover community detection approaches can be classified in two categories: global, which require the knowledge of entire network, and local algorithms, which expand an initial seed set of nodes into possibly overlapping communities by examining only a small part of the network.

Finally, the algorithms for community detection can be classified as in the following: (a) cohesive subgraph discovery, (b) vertex clustering, (c) community quality optimization, (d) divisive, and (e) model based.

References

- Amato F, Moscato V, Picariello A, Piccialli F, Sperlí G (2016a) Centrality in heterogeneous social networks for lurkers detection: an approach based on hypergraphs. *Concurr Comput Pract Exp* 30(3):e4188
- Amato F, Moscato V, Picariello A, Sperlí G (2016b) Multimedia social network modeling: a proposal. In: 2016 IEEE tenth international conference on semantic

- computing (ICSC), pp 448–453. <https://doi.org/10.1109/ICSC.2016.20>
- Amato F, Moscato V, Picariello A, Sperli G (2017a) Diffusion algorithms in multimedia social networks: a preliminary model. In: Proceedings of SocialInfluence@ASONAM 2017
- Amato F, Moscato V, Picariello A, Sperli G (2017b) Recommendation in social media networks. In: Third IEEE international conference on multimedia big data, BigMM 2017, Laguna Hills, 19–21 Apr 2017, pp 213–216. <https://doi.org/10.1109/BiMM.2017.55>
- Bucur D, Iacca G (2016) Influence maximization in social networks with genetic algorithms. In: European conference on the applications of evolutionary computation. Springer, pp 379–392
- Chakraborty T, Dalmia A, Mukherjee A, Ganguly N (2017) Metrics for community analysis: a survey. ACM Comput Surv (CSUR) 50(4):54
- Fortunato S (2010) Community detection in graphs. Phys Rep 486(3):75–174
- Goldenberg J, Libai B, Muller E (2001) Talk of the network: a complex systems look at the underlying process of word-of-mouth. Mark Lett 12(3):211–223
- Gulbahce N, Lehmann S (2008) The art of community detection. BioEssays 30(10):934–938
- Keeling M, Eames K (2005) Networks and epidemic models. J R Soc Interface 2:295–309
- Kempe D, Kleinberg J, Tardos E (2003a) Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, KDD'03. ACM, New York, pp 137–146. <http://doi.acm.org/10.1145/956750.956769>
- Kempe D, Kleinberg J, Tardos É (2003b) Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 137–146
- Papadopoulos S, Kompatsiaris Y, Vakali A, Spyridonos P (2012) Community detection in social media. Data Min Knowl Disc 24(3):515–554
- Porter MA, Onnela JP, Mucha PJ (2009) Communities in networks. Not AMS 56(9):1082–1097
- Sankar CP, Ashraf S, Kumar KS (2016) Learning from bees: an approach for influence maximization on viral campaigns. PLoS One 11(12):e0168125
- Tan W, Brian M, Salehi I, Durstdar S (2013) Social-network-sourced big data analytics. Internet Comput J 17:62–69
- Tang Y, Xiao X, Shi Y (2014) Influence maximization: near-optimal time complexity meets practical efficiency. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 75–86
- Tang Y, Shi Y, Xiao X (2015) Influence maximization in near-linear time: a martingale approach. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 1539–1554
- Vaswani S, Lakshmanan L, Schmidt M et al (2015) Influence maximization with bandits. arXiv preprint arXiv:150300024
- Wang M, Wang C, Yu JX, Zhang J (2015) Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. Proc VLDB Endow 8(10):998–1009
- Yang B, Liu D, Liu J (2010) Discovering communities from social networks: methodologies and applications. In: Furht B (ed) Handbook of social network technologies and applications. Springer, pp 331–346

Big Data in the Cloud

S. M. Zobaed and Mohsen Amini Salehi
High Performance Cloud Computing (HPCC)
laboratory, University of Louisiana at Lafayette,
Lafayette, LA, USA

Definitions

Then eventually discuss big data handling challenges, issues, and how big data can be stored, processed, and accessed in the cloud.

Overview

Cloud storage services have emerged to address the increasing demand to store and process huge amount of data, generally alluded as “Big Data” (Wu et al. 2014). Typically, organizations store the huge volume of data to various clouds.

Cloud computing offers organizations the ability to manage big data and process them without the cost and burden of maintaining and upgrading local computing resources. However, efficient utilization of clouds for big data imposes new challenges in several domains. In this chapter, we discuss challenges in big data storage, distribution, security, and real-time processing. It is also explained how clouds can be instrumental for big data generated by Internet of Things (IoT). An overview of popular tools that are available in clouds for big data analytics is depicted. Finally, there is a discussion on future research directions in these areas.

Key Research Findings Big Data in the Cloud

Storage Levels for Big Data in the Cloud

A cloud storage consists of enormous number (order of thousands) of storage server clusters connected by a high-bandwidth network. A storage middleware (e.g., SafeSky (Zhao et al. 2015)) is used to provide distributed file system and to deal with storage allocation throughout the cloud storage.

Generally, cloud providers offer various storage levels with different pricing and latencies. These storage levels can be utilized to store big data in the cloud, depending on the price and latency constraints. Amazon cloud, for instance, provides *object storage*, *file storage*, and *block storage* to address the sheer size, velocity, and formats of big data.

Object storage is cheaper and slower to access, generally used to store large objects with low access rate (e.g., backups and archive data). This type of storage service operates based on Hard Disk Drive (HDD) technology. Amazon simple storage service (<https://aws.amazon.com/s3/>) (S3) is an example of Object storage service (Wu et al. 2010).

File storage service is also offered based on HDD storage technology. However, it provides file system interface, file system access consistency, and file locking. Unlike object storage, file storage capacity is elastic which means growing and shrinking automatically in response to addition and removal of data. Moreover, it works as a concurrently accessible storage for up to thousands of instances. Amazon Elastic File System (<https://aws.amazon.com/efs/>) (EFS) is an example of this type of storage.

Block storage level is offered based on Solid-State Drives (SSD) and provides ultra-low access latency. Big data with frequent access or delay sensitive can be stored in this storage level. Amazon Elastic Block Store (EBS) is an example of Amazon cloud service offered based on block storage. High-performance big data applications (e.g., Sagiroglu and Sinanc 2013) commonly use storage level to minimize their access latency.

Cloud Big Data: Persistence Versus Distribution

Although clouds provide an ideal environment for big data storage, the access delay to them is generally high due to network limitations (Terzo et al. 2013). The delay can be particularly problematic for frequently accessed data (e.g., index of big data search engine). As such, in addition to storage services, other cloud services need to be in place to manage distribution of big data so that the access delay can be reduced. That is, separating the big data storage in the cloud from the distribution.

Content Delivery Network (CDN) is a network of distributed servers aiming to reduce data access delay over the Internet (Pierre and Van Steen 2006). The CDN replicates and caches data within a network of servers dispersed at different geographical locations.

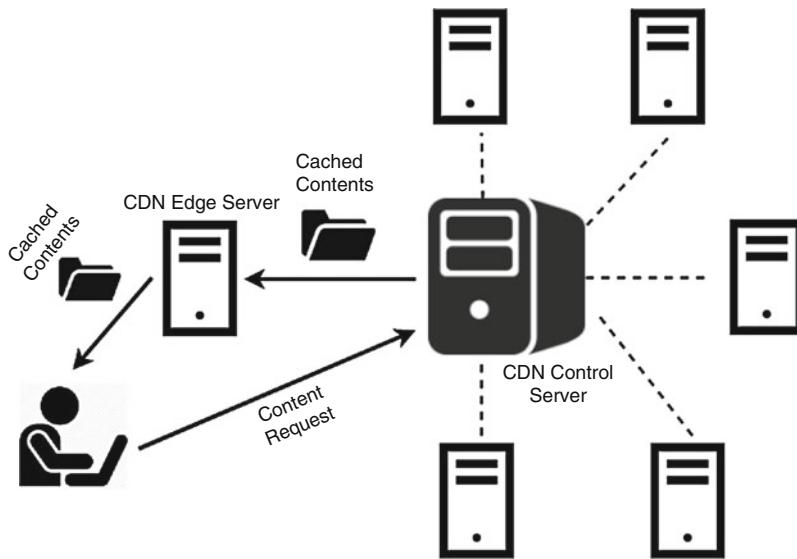
Figure 1 depicts how CDN technology operates. As we can see in this figure, upon receiving a request to access data, CDN control server caches the data in the nearest edge server to deliver it with minimum latency.

Cloud providers offer distribution services through built-in CDN services or by integrating with current CDN providers. For instance, Amazon cloud offers CloudFront (Dignan 2008) CDN service. Akamai, a major CDN provider has been integrated with Microsoft Azure cloud to provide short delay in accessing big data (Sirivara 2016).

Real-Time Big Data Analytics in the Cloud

Cloud big data can be subject to real-time processing in applications that detect patterns or perceive insights from big data. For instance, in S3C (Woodworth et al. 2016) huge index structures need to be accessed to enable a real-time search service. Other instances are big data analytics for sensor data (Hu et al. 2014), financial data (Hu et al. 2014), and streaming data (Li et al. 2016).

The faster organizations can fetch insights from big data, the greater chance in generating revenue, reducing expenditures, and increasing productivity. The main advantage of separating



Big Data in the Cloud, Fig. 1 CDN technology helps to minimize access latency for frequently accessed big data in the cloud

storage from distribution is to help organizations to process big data in real-time.

In most cases, data collected from various sources (e.g., sensor data, organizations' data) are raw and noisy, thus, are not ready for analysis. Jagannathan (2016) propose a method to deal with this challenge.

Security of Big Data in the Cloud

One major challenge in utilizing cloud for big data is security concerns (Dewangan and Verma 2015). The concern for big data owners is essentially not having full control over their data hosted by clouds. The data can be exposed to external or, more importantly, to internal attacks.

A survey carried out by McKinsey in 2016 (Elumalai et al. 2016) showed that security and compliance are the two primary considerations in selecting a cloud provider, even beyond cost benefits. In October 2016, external attackers performed a Distributed Denial of Service (DDoS) attack that made several big data dealing organizations (e.g., Twitter, Spotify, Github) inaccessible. Prism (Greenwald and MacAskill 2013) is an example of an internal attack launched in 2013. In this section, we discuss major security challenges for big data in the cloud and current efforts to address them.

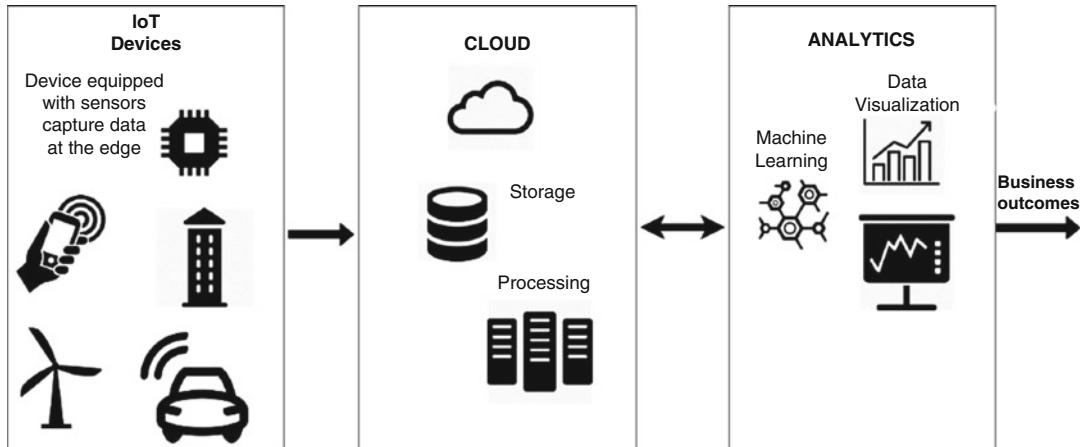
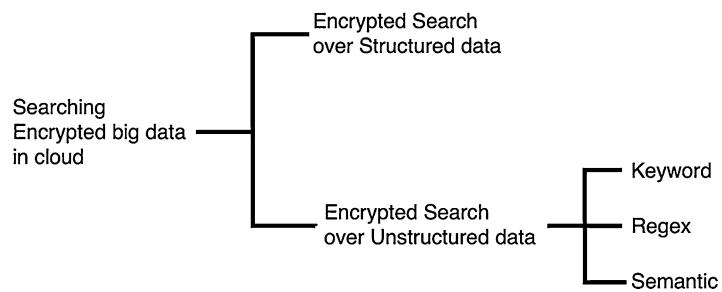
Security of Big Data Against Internal Attackers

In addition to common security measures (e.g., logging or node maintenance against malware), user-side encryption (Salehi et al. 2014; Woodworth et al. 2016) is becoming a popular approach to preserve the privacy of big data hosted in the cloud. That is, the data encryption is done using user's keys and in the user premises; therefore, external attackers cannot see or tamper with the data. However, user-side encryption brings about higher computational complexity and hinders searching big data (Wang et al. 2010).

Several research works have been undertaken recently to initiate different types of search over encrypted data in the cloud. Figure 2 provides a taxonomy of research works on the search over encrypted data in the cloud. In general, proposed search methods are categorized as search over structured and unstructured data. Unstructured searches can be further categorized as keyword search, Regular expression search, and semantic search.

Privacy-Preserving query over encrypted graphstructured data (Cao et al. 2011) is an instance of search over encrypted structured data. REseED (Salehi et al. 2014), SSE (Curtmola et al. 2006), and S3C (Woodworth et al. 2016) are

Big Data in the Cloud,
Fig. 2 Taxonomy of different types of search over encrypted big data



Big Data in the Cloud, Fig. 3 Work-flow of IoT systems dealing with big data in the cloud

tools developed for regular expression (Regex), keyword, and semantic searching, respectively, over unstructured big data in the cloud. Big data integrity in the cloud is another security concern. The challenge is how to verify the data integrity in the cloud storage without downloading and then uploading the data back to cloud. Rapid incoming of new data to the cloud makes verifying data integrity further complicated (Chen and Zhao 2012). Techniques such as continuous integrity monitoring are applied to deal with big data integrity challenge in the cloud (Lebdaoui et al. 2016).

Security Against External Attackers

External attackers are generally considered as potential threats to the cloud that handle big data. If cloud storages face external attacks, users' data will be leaked and controlled by attackers. So, implementing security on the cloud that handles big data is a challenge.

To address the security challenge, researches have been done in that area. For instance, a layered framework (Reddy et al. 2012) is proposed for assuring big data analytics security in the cloud. It consists of securing multiple layers, namely, virtual machine layer, storage layer, cloud data layer, and secured virtual network monitor layer.

IoT Big Data in the Cloud

The Internet of Things (IoT) is a mesh of physical devices embedded with electronics, sensors, and network connectivity to collect and exchange data (Atzori et al. 2010). IoT devices are deployed in different systems, such as smart cities, smart homes, or to monitor environments (e.g., smart grid). Basic work-flow of IoT systems that use cloud services (Dolgov 2017) is depicted in Fig. 3. According to the figure, the IoT devices produce streams of big data that are hosted and analyzed using cloud services. Many of these

systems (e.g., monitoring system) require low latency (real-time) analytics of the collected data.

Big data generated in IoT systems generally suffer from redundant or noisy data which can affect big data analytic tools (Chen et al. 2014).

Knowledge Discovery in Databases (KDD) and data mining technologies offer solutions for analysis of big data generated by IoT systems (Tsai et al. 2014). The big data are converted into useful information using KDD. The output of KDD is further processed using data mining or visualization techniques to extract insights or patterns of the data.

Amazon IOT (<https://aws.amazon.com/iot/>), Dell Statistica (<http://iotworm.com/internet-things-business-data-analytics-tools/>), Amazon Greengrass (<https://aws.amazon.com/greengrass/>) Azure stream analytics are tools that function based on the aforementioned work-flow in clouds to extract insights from IoT big data with low latency.

Fog computing (Bonomi et al. 2012) extends the ability of clouds to edge servers with the goal of minimizing latency and maximizing the efficiency of core clouds. Fog computing can also be used to deal with noisy big data in IoT (Bonomi et al. 2012). For instance, Amazon service enables creation of fog computing that can be pipelined to other Amazon cloud services (e.g., Amazon SimpleDB and EMR) for big data processing.

Cloud-Based Big Data Analytics Tools

Unlike conventional data that are either structured (e.g., XML, XAML, and relational databases) or unstructured, big data can be a mixture of structured, semi-structured, or unstructured data (Wu et al. 2014). It is approximated that around 85% of total data come from organization are unstructured and moreover, almost all the individuals generated data are also unstructured (Pusala et al. 2016).

Such heterogeneous big data cannot be analyzed using traditional database management tools (e.g., relational database management system (RDBMS)). Rather, a set of new cloud-based processing tools have been developed for processing big data (Chen et al. 2014).

NoSQL-NoSQL (generally referred to as “Not Only SQL”) is a framework for databases that provides high-performance processing for big data. In fact, conventional relational database systems have several features (e.g., transactions management and concurrency control) that make them unscalable for big data. In contrast, NoSQL databases are relieved from these extra features and lend themselves well to distributed processing in the clouds. These database systems can generally handle big data processing in real-time or near real-time. In this part, an overview of the NoSQL databases offered by clouds is explained.

- **Hadoop MapReduce** – MapReduce framework was published by google in 2005 (Ditttrich and Quiané-Ruiz 2012). Hadoop is an open source implementation tool based on MapReduce framework developed by Apache (Vavilapalli et al. 2013). Hadoop is a batch data analytics technology that is designed for failure-prone computing environments. The specialty of Hadoop is that developers need to define only the map and reduce functions that operate on big data to achieve data analytics. Hadoop utilizes a special distributed file system, named Hadoop Distributed File System (HDFS), to handle data (e.g., read and write operations). Hadoop achieves fault-tolerance through data replication. Amazon Elastic MapReduce (EMR) (<https://aws.amazon.com/emr/details/hadoop/>) is an example of Hadoop framework (Jourdren et al. 2012) offered as a service by Amazon cloud. Amazon EMR helps to create and handle elastic clusters of Amazon EC2 instances running Hadoop and other applications in the Hadoop system.

- **Key-Value Store** – key-value database system is designed for storing, retrieving, and handling data in mapping format where data are considered as a value and a key is used as an identifier of a particular data as because keys and values are resided pairwise in the database system. Amazon DynamoDB (<https://aws.amazon.com/dynamodb/>), GoogleBigTable (<https://cloud.google.com/bigtable/>), and HBase (<https://hbase.apache.org/>) are examples of cloud-based key-value store databases to handle big data.

• **Document Store** – These database systems are similar to key-value store databases. However, keys are mapped to documents, instead of values. Pairs of keys and documents (e.g., in JASON or XML formats) are used to build data model over big data in the cloud. This type of database systems are used to store semi-structured data as documents, usually in JSON, XML, or XAML formats. Amazon cloud offers Amazon SimpleDB (<https://aws.amazon.com/simpledb/>) as a document store database service that creates and manages multiple distributed replicas of data automatically to enable high availability and data durability. Data owner can change data model on the fly, and data are automatically indexed later on.

• **Graph Databases** – These database systems are used for processing data that have graph nature. For instance, users' connections in a social network can be modeled using a graph database (Shimpi and Chaudhari 2012). In this case, users are considered as nodes and connections between users are represented as edges in the graph. The graph can be processed to predict links between nodes. That is, possible connections between users. Giraph (<http://giraph.apache.org/>), Neo4j (<https://zeroturnaround.com/rebellabs/examples-where-graph-databases-shine-neo4j-edition/>), and FlockDB (https://blog.twitter.com/engineering/en_us/a/2010/introducing-flockdb.html) are examples of cloud-based graph database services. Facebook (<https://www.facebook.com>), a large social networking site, is currently using Giraph database system to store and analyze their users' connections.

In Memory Analytics – These are techniques that process big data in the main memory (RAM) with reduced latency (Dittrich and Quiané-Ruiz 2012). For instance, Amazon provides Elastic-Cache (<https://aws.amazon.com/elasticcache/>) service to improve web applications' performance which at previous generally relied on slower disk-based databases. Beside that there are also tools for performing **in memory analytics** on the big data in cloud such as Apache Spark, SAP, and Microstrategy (Sultan 2015).

Different big data analytic tools introduced in this section are appropriate for different types of datasets and applications. There is no single perfect solution for all types of big data analytics in the cloud.

Future Directions for Research

Cloud-based big data analytics has progressed significantly over the past few years. The progression has led to emergence of new research areas that need further investigations from industry and academia. In this section, an overview of these potential areas is provided.

Different businesses use various cloud providers to handle their big data. However, current big data analytics tools are limited to a single cloud provider. Solutions are required to bridge big data stored in different clouds. Such solutions should be able to provide real-time processing of big data across multiple clouds. In addition, such solutions should consider security constraints of big data reside in different clouds.

The distribution of access to data is not uniform in all big datasets. For instance, social network pages and video stream repositories (Li et al. 2016 and Darwich et al. 2017) have long-tail access patterns. That is, only few portions of big data are accessed frequently, while the rest of the datasets are rarely accessed. Solutions are required to optimally place big data in different cloud storage types, with respect to access rate to the data. Such solutions can minimize the incurred cost of using cloud and access latency to the data.

Security of big data in cloud is still a major concern for many businesses (Woodworth et al. 2016). Although there are tools that enable processing (e.g., search) of user-side encrypted big data in cloud, further research is required to expand processing of such data without revealing them to the cloud. Making use of homomorphic encryption (Naehrig et al. 2011) is still in its infancy. It can be leveraged to enable more sophisticated (e.g., mathematical) data processing without decrypting data in the cloud.

Conclusion

The size of digital data is rapidly growing to the extent that storage and processing have become challenging. Cloud services have emerged to address these challenges. Nowadays, cloud providers offer collection of services that can address different big data demands. In particular, they provide levels of storage with different prices and access latencies. They also provide big data distribution based on CDN techniques to reduce access latency further. For big data analytics, clouds offer several services, including NoSQL databases and in memory big data analytics.

Although clouds have been instrumental in enabling big data analytics, security and privacy of data are still major impediments for many businesses to embrace clouds.

References

- Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. *J Comput Netw* 54(15): 2787–2805
- Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the 1st edition of the MCC workshop on mobile cloud computing, MCC'12, Bonomi- Helsinki, Finland, pp 13–16
- Cao N, Yang Z, Wang C, Ren K, Lou W (2011) Privacy-preserving query over encrypted graph-structured data in cloud computing. In: Proceedings of the 31st international conference on distributed computing systems. ICDCS'11. Washington, DC, pp 393–402. ISBN: 978-0-7695-4364-2
- Chen D, Zhao H (2012) Data security and privacy protection issues in cloud computing. In: Proceedings of international conference on computer science and electronics engineering, vol 1. ICC-SEE'12, pp 647–651
- Chen M, Mao S, Liu Y (2014) Big data: a survey. *J Mob Netw Appl* 19(2):171–209
- Curtmola R, Garay J, Kamara S, Ostrovsky R (2006) Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on computer and communications security. CCS'06, Virginia, USA, pp 79–88
- Darwich M, Beyazit E, Salehi MA, Bayoumi M (2017) Cost Efficient Repository Management for Cloud-Based On Demand Video Streaming. In: Proceedings of the 5th international conference on mobile cloud computing, services, and engineering. IEEE mobile cloud'17. San Francisco
- Dewanjan AK, Verma G (2015) A security mechanism for cloud computing threats. *Int J Comput Appl Comput Electron Welf Rural Masses* 1:18
- Dignan L (2008) Amazon launches CloudFront; Content delivery network margins go kaboom. <http://www.zdnet.com/article/amazon-launches-cloudfront-content-delivery-network-margins-go-kaboom/>. Online; Accessed 13 Oct 2017
- Dittrich J, Quiané-Ruiz J-A (2012) Efficient big data processing in Hadoop MapReduce. *J VLDB Endowment* 5(12):2014–2015
- Dolgov S (2017) AI marketplace: neural network in your shopping cart. <https://www.linkedin.com/pulse/ai-marketplace-neural-network-your-shopping-cart-sergey-dolgov-1/>. Online; Accessed 13 Oct 2017
- Elumalai A, Starikova I, Tandon S (2016) IT as a service: from build to consume. <https://www.mckinsey.com/industries/high-tech/our-insights/it-as-a-service-from-build-to-consume/>. Online; Accessed 12 Oct 2017
- Greenwald G, MacAskill E (2013) NSA Prism program taps in to user data of Apple, Google and others. *J Guardian* 7(6):1–43
- Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. *J IEEE Access* 2:652–687
- Jagannathan S (2016) Real-time big data analytics architecture for remote sensing application. In: Proceedings of the 19th international workshop on software and compilers for embedded systems international conference on signal processing, communication, power and embedded system. SCOPES'16, Germany, pp 1912–1916
- Jourdren L, Bernard M, Dillies M-A, Le Crom S (2012) Eoulsan. *J Bioinforma* 28(11):1542–1543
- Lebdaoui I, El Hajji S, Orhanou G (2016) Managing big data integrity. In: Proceedings of international conference on engineering & MIS. ICEMIS'16, Agadir, Morocco, pp 1–6
- Li X, Salehi MA, Bayoumi M, Buyya R (2016) CVSS: a cost-efficient and QoS-aware video streaming using cloud services. In: Proceedings of the 16th IEEE/ACM international symposium on cluster, cloud and grid computing. CCGrid'16. IEEE, Cartagena, Colombia, pp 106–115
- Naehrig M, Lauter K, Vaikuntanathan V (2011) Can homomorphic encryption Be practical? In: Proceedings of the 3rd ACM workshop on cloud computing security workshop. CCSW'11. Chicago, pp 113–124
- Pierre G, Van Steen M (2006) Globule: a collaborative content delivery network. *J Commun Mag* 44(8): 127–133
- Pusala MK, Salehi MA, Katukuri JR, Xie Y, Raghavan V (2016) Massive data analysis: tasks, tools, applications, and challenges. In: Big data analytics. Springer, New Delhi, India, pp 11–40
- Reddy CKK, Anisha PR, Srinivasulu Reddy K, Surender Reddy S (2012) Third party data protection applied

- to cloud and XACML implementation in the hadoop environment with sparql. *J Int Organ Sci Res Comput Eng* 2(1):39–46. ISSN: 2278-0661, New Delhi, India
- Sagiroglu, Sinanc (2013) Big data: a review. In: Proceedings of international conference on collaboration technologies and systems. CTS'13, San Diego, California, USA, pp 42–47
- Salehi MA, Caldwell T, Fernandez A, Mickiewicz E, Rozier EWD, Zonouz S, Redberg D (2014) RE-SeED: regular expression search over encrypted data in the cloud. In: Proceedings of the 7th international conference on cloud computing. CLOUD'14, pp 673–680
- Shimpi D, Chaudhari S (2012) An overview of graph databases. In: Proceedings of the 2nd international conference in recent trends in information technology and computer science. ICRTITCS'12, India, pp 16–22
- Sirivara S (2016) Windows Azure content delivery network. <https://azure.microsoft.com/en-us/blog/azure-cdn-from-akamai-ga/>. Online; Accessed 13 Oct 2017
- Sultan (2015) Top 10 in-memory business intelligence analytics tools. <https://www.mytechlogy.com/IT-blogs/9507/top-10-in-memory-business-intelligence-analytics-tools/>. Online; Accessed 12 Oct 2017
- Terzo O, Ruiu P, Bucci E, Xhafa F (2013) Data as a service (DaaS) for sharing and processing of large data collections in the cloud. In: Proceedings of the 7th international conference on complex, intelligent, and software intensive systems. CISIS'2013, Taichung, Taiwan, pp 475–480
- Tsai C-W, Lai C-F, Chiang M-C, Yang LT et al (2014) Data mining for internet of things: a survey. *J IEEE Commun Surv Tutorials* 16(1):77–97
- Vavilapalli VK et al (2013).Apache hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. SOCC'13, New York, pp 5:1–5:16. ISBN: 978-1-4503-2428-1
- Wang C, Cao N, Li J, Ren K, Lou W (2010) Secure ranked keyword search over encrypted cloud data. In: Proceedings of the 30th international conference on distributed computing systems. ICDCS'10, Genoa, Italy, pp 253–262
- Woodworth J, Salehi MA, Raghavan V (2016) S3C: an architecture for spaceefficient semantic search over encrypted data in the cloud. In: Proceedings of international conference on Big data. Big Data'16, Washington DC, pp 3722–3731
- Wu J, Ping L, Ge X, Wang Y, Jianqing Fu (2010) Cloud storage as the infrastructure of cloud computing. In: Proceeding of 9th international conference on intelligent computing and cognitive informatics. ICICCI'10, Kuala Lumpur, Malaysia, pp 380–383
- Wu X, Zhu X, Wu G-Q, Ding W (2014) Data mining with big data. *J IEEE Trans Knowl Data Eng* 26(1):97–107
- Zhao R, Yue C, Tak B, Tang C (2015) SafeSky: a secure cloud storage middleware for end-user applications. In: Proceedings of the 34th IEEE symposium on reliable distributed systems. SRDS'15, Montreal, QC, Canada, pp 21–30

Big Data Indexing

Mohamed Y. Eltabakh

Worcester Polytechnic Institute, Worcester, MA, USA

B

Definitions

The major theme of this topic is building indexes, which are auxiliary data structures, on top of big datasets to speed up its retrieval and querying. The topic covers a wide range of index types along with a comparison of their structures and capabilities.

Overview

Big data infrastructures such as Hadoop are increasingly supporting applications that manage structured or semi-structured data. In many applications including scientific applications, weblog analysis, click streams, transaction logs, and airline analytics, at least partial knowledge about the data structure is known. For example, some attributes (columns in the data) may have known data types and possible domain of values, while other attributes may have little information known about them. This knowledge, even if it is partial, can enable optimization techniques that otherwise would not be possible.

Query optimization is a core mechanism in data management systems. It enables executing users' queries efficiently without the users having to know how their queries will execute. A query optimizer figures out the best query plan to execute, in which order the query operators will run, where the data should be located and how will it move during the processing, and which segments of a query can execute in parallel versus running in sequence. Query optimization in big data is highly important, especially because (1) the datasets to be processed are getting very large, (2) the analytical queries are increasing in complexity and may take hours to execute if not carefully optimized, and (3) the pay-as-you-go

cost models for cloud computing add additional urgency for optimized processing.

A typical query in big data applications may touch files in the order of 100s of GBs or TBs of size. These queries are typically very expensive as they consume significant resources and require long periods of time to execute. For example, in transaction log applications, e.g., transaction history of customer purchases, one query might be interested in retrieving all transactions from the last 2 months that exceed a certain amount of dollar money. Such query may need to scan billions of records and go over TBs of data.

Indexing techniques are well-known techniques in database systems, especially relational databases (Chamberlin et al. 1974; Maier 1983; Stonebraker et al. 1990), to optimize query processing. Examples of the standard indexing techniques are the B+-tree (Bayer and McCreight 1972), R-Tree (Guttman 1984), and Hash-based indexes (Moro et al. 2009) along with their variations. However, transforming these techniques and structures to big data infrastructures is not straightforward due to the unique characteristics of both the data itself and the underlying infrastructure processing the data. At the data level, the data is no longer assumed to be stored in relational tables. Instead, the data is received and stored in the forms of big batches of flat files. In addition, the data size exceeds what relational database systems can typically handle.

On the other hand, at the infrastructure level, the processing model no longer follows the relational model of query execution, which relies on connecting a set of query operators together to form a query tree. Instead, the MapReduce computing paradigm is entirely different as it relies on two rigid phases of *map* and *reduce* (Dean and Ghemawat 2008). Moreover, the access pattern of the data from the file system is also different. In relational databases, the data records are read in the form of disk pages (a.k.a disk blocks), which are very small in size (typically between 8 to 128 KBs) and usually hold few data records (10s or at most 100s of records). And thus, it is assumed that the database systems can support record-level access. In contrast, in the Hadoop file system (HDFS), a single data block ranges

between 64 MBs to 1 GB, and usually holds many records. Therefore, the record-level access no longer holds. Even the feasible operations over the data are different from those supported in relational databases. For example, record updates and deletes are not allowed in the MapReduce infrastructure. All of these unique characteristics of big data fundamentally affect the design of the appropriate indexing and preprocessing techniques.

Plain Hadoop is found to be orders of magnitude slower than distributed database management systems when evaluating queries on structured data (Stonebraker et al. 2010; Abadi 2010). One of the main observed reasons for this slow performance is the lack of indexing in the Hadoop infrastructure. As a result, significant research efforts have been dedicated to designing indexing techniques suitable for the Hadoop infrastructure. These techniques have ranged from record-level indexing (Dittrich et al. 2010, 2012; Jiang et al. 2010; Richter et al. 2012) to split-level indexing (Eltabakh et al. 2013; Gankidi et al. 2014), from user-defined indexes (Dittrich et al. 2010; Jiang et al. 2010; Gankidi et al. 2014) to system-generated and adaptive indexes (Richter et al. 2012; Dittrich et al. 2012; Eltabakh et al. 2013), and from single-dimension indexes (Dittrich et al. 2010, 2012; Jiang et al. 2010; Richter et al. 2012; Eltabakh et al. 2013) to multidimensional indexes (Liu et al. 2014; Eldawy and Mokbel 2015; Lu et al. 2014).

Table 1 provides a comparison among several of the Hadoop-based indexing techniques with respect to different criteria. Record-level granularity techniques aim for skipping irrelevant records within each data split, but eventually they may touch all splits. In contrast, the split-level granularity techniques aim for skipping entire irrelevant splits. SpatialHadoop system provides both split-level global indexing and record-level local indexing, and thus it can skip irrelevant data at both granularities.

Some techniques index only one attribute at a time (Dimensionality = 1), while others allow indexing multidimensional data (Dimensionality = m). Techniques like HadoopDB and Polybase Index inherit the multidimensional capabilities

Big Data Indexing, Table 1 Comparison of hadoop-based indexed techniques

Technique	Granularity	Dimensionality	DB-Hybrid	Definition	Index Location
Hadoop++ (Dittrich et al. 2010)	Record	1	No	Admin	HDFS
HAIL (Dittrich et al. 2012)	Record	1	No	Admin	HDFS
LIAH (Richter et al. 2012)	Record	1	No	System	HDFS
E3 (Eltabakh et al. 2013)	Split	1 & 2	No	System	DB
SpatialHadoop (Eldawy and Mokbel 2015)	Record/Split	m	No	Admin	HDFS
ScalaGist (Lu et al. 2014)	Record	m	No	Admin	HDFS
HadoopDB (Abouzeid et al. 2009)	Record	m	Yes	Admin	DB
Polybase Index (Gankidi et al. 2014)	Split	m	Yes	Admin	DB

from the underlying DBMS. E3 technique enables indexing pairs of values (from two attributes) but only for a limited subset of the possible values. Most techniques operate only on the HDFS data (DB-Hybrid = N), while HadoopDB and Polybase Index have a database system integrated with HDFS to form a hybrid system.

In most of the proposed techniques, the system's admin decides on which attributes to be indexed. The only exceptions are the LIAH index, which is an adaptive index that automatically detects the changes of the workload and accordingly creates (or deletes) indexes, and the E3 index, which automatically indexes all attributes in possibly different ways depending on the data types and the workload. Finally, the index structure is either stored in HDFS along with its data as in Hadoop++, HAIL, LIAH, SpatialHadoop, and ScalaGist, in a database system along with its data as in HadoopDB, or in a database system while the data resides in HDFS as in E3 and Polybase Index. The following sections cover few of these techniques in more details.

Target Queries: Indexing techniques target optimizing queries that involve selection predicates, which is the common theme for all techniques listed in Table 1. Yet, they may differ on how queries are expressed and the mechanism by which the selection predicates are identified. For example, Hadoop++, HAIL, and LIAH allow expressing the query in Java while passing the selection predicates as arguments within the map-reduce job configuration. As a result, a customized input format will receive these predicates (if any) and perform the desired filtering during execution. In contrast, E3 framework is built on

top of the Jaql high-level query language (Beyer et al. 2011), and thus queries are expressed, compiled, and optimized using Jaql engine (Beyer et al. 2011). An example query is as follows:

```
read( hdfs("docs.json") )
-> transform {
author: $.meta.author,
products: $.meta.product,
Total: $.meta.Qty * $.meta.
    Price}
-> filter $.products == "XYZ";
```

Jaql has the feature of applying *selection-push-down* during query compilation whenever possible. As a result, in the given query the filter operator will be pushed before the transform operator, with the appropriate rewriting. The E3 framework can then detect this filtering operation directly after the read operation of the base file and thus can push the selection predicate into its customized input format to apply the filtering as early as possible.

HadoopDB provides a front-end for expressing SQL queries on top of its data, which is called SMS. SMS is an extension to Hive. In HadoopDB queries are expressed in an identical way to standard SQL as in the following example:

```
SELECT pageURL, pageRank
  FROM Rankings
 WHERE pageRank > 10;
```

SpatialHadoop is designed for spatial queries, and thus it provides a high-level language and constructs for expressing these queries and operating on spatial objects, e.g., points and

rectangles. For example, a query can be expressed as follows:

```
Objects = LOAD "points" AS
(id:int, Location:POINT);
Result = FILTER Objects BY
Overlaps (Location,
Rectangle(x1, y1, x2, y2));
```

ScalaGist enables building Gist indexes, e.g., B+-tree and R-tree, over HDFS data. A single query in ScalaGist can make use of multiple indexes at the same time. For example, given a table T with schema $\{x, \dots, (a1, a2)\}$, where x is a one-dimensional column and $(a1, a2)$ is a two-dimensional column, the following query can use both a B+-tree index (on x) and an R-tree index (on $(a1, a2)$) during its evaluation:

```
SELECT *
FROM T
WHERE x ≤ 100
    AND 10 ≤ a1 ≤ 20
    AND 30 ≤ a2 ≤ 60;
```

Finally, the Polybase system enables expressing queries using standard SQL over HDFS data that are defined as external tables. First, users need to define the external table as in the following example:

```
Create External Table
hdfsLineItem
(l_orderkey BIGINT Not Null,
l_partkey BIGINT Not Null,
...)
With (Location = '/tpch1gb/
lineitem.tbl',
Data_Source = VLDB_HDP_Cluster,
File_Format = Text_Delimited);
```

And then, a query on the external table can be expressed as follows:

```
SELECT *
FROM hdfsLineItem
WHERE l_orderkey = 1;
```

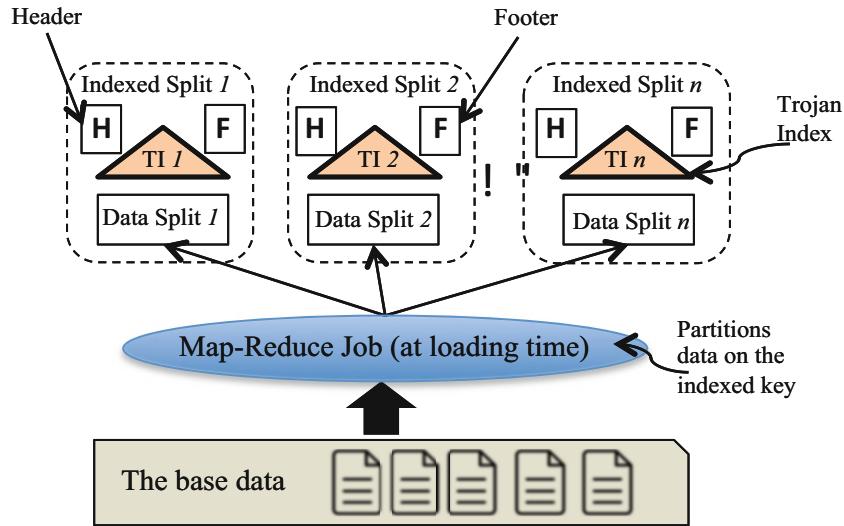
Record-Level Nonadaptive Indexing

Hadoop++ (Dittrich et al. 2010) is an indexing technique built on top of the Hadoop infrastructure. Unlike other techniques that require extensive changes to Hadoop's execution model to offer run-time optimizations, e.g., HadoopDB (Abouzeid et al. 2009; Abouzied et al. 2010), Hadoop++ relies on augmenting the indexing structures to the data in a way that does not affect the execution mechanism of Hadoop. All processing on the indexes, e.g., creating the indexes, augmenting them to the data, and their access, are all performed through pluggable user-defined functions (UDFs) that are already available within the Hadoop framework.

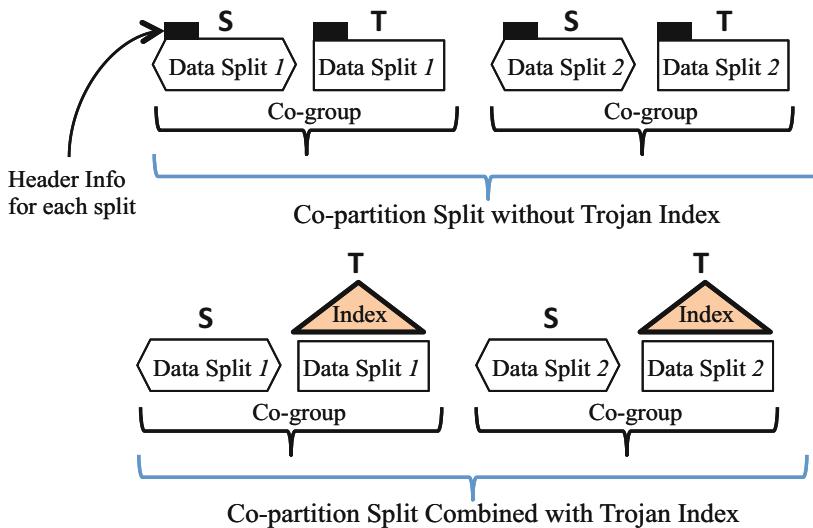
The basic idea of the Hadoop++ index, which is referred to as *a trojan index*, is illustrated in Fig. 1a. At the loading time, the base data is partitioned using a map-reduce job. This job partitions the data based on the attribute to be indexed, i.e., if attribute X is to be indexed, then depending on the X 's value in each record, the record will be assigned to a specific split Id. This assignment is performed by the mapper function. On the other hand, the reducer function receives all the records belonging to a specific split and creates the trojan index corresponding to that split.

The index is then augmented to the data split to form a bigger split, referred to as *an indexed split* as depicted in the figure. Each indexed split will also have a *split header (H)*, and a *split footer (F)*, which together hold the metadata information about each indexed split, e.g., the split size, the number of records, the smallest and largest indexed values within this split, etc. In general, Hadoop++ can be configured to create several trojan indexes for the same data on different attributes. However, only one index can be the primary index according to which the data records are sorted within each split. This primary index is referred to as the *clustered index*, while the other additional indexes are *non-clustered indexes*.

At query time, given a query involving a selection predicate on one of the indexed attributes, the processing works as follows. First, a custom InputFormat function would read each indexed split



(a) Hadoop++ Trojan Index



(b) Hadoop++ Trojan Join

Big Data Indexing, Fig. 1 Hadoop++ Trojan Index and Trojan Join

(instead of the data splits) and consult the trojan index for that split w.r.t the selection predicate. If there are multiple indexes, then the appropriate index is selected based on the selection predicate. If none of the records satisfies the query predicate, then the entire split is skipped, and the map function terminates without actually checking any record within this split. Otherwise, the

trojan index will point to the data records within the split that satisfies the query. If the trojan index is clustered, then this means that the data records within the given block are ordered according to the indexed attribute, and thus the retrieval of the records will be faster and requires less I/Os.

It is worth highlighting that trojan indexes are categorized as *local* indexes meaning that a local

index is created for each data split in contrast to building a single global index for the entire dataset. Local indexes have their advantages and disadvantages. For example, one of the advantages is that the entire dataset does not need to be sorted, which is important because global sorting is prohibitively expensive in big data. However, one disadvantage is that each indexed split has to be touched at query time. This implies that a mapper function has to be scheduled and initiated by Hadoop for each split even if many of these splits are irrelevant to the query.

Hadoop++ framework also provides a mechanism, called *Trojan Join*, to speed up the join operation between two datasets, say S and T (refer to Fig. 1b). The basic idea is to partition both datasets (at the same time) on the join key. This partitioning can be performed at the loading time as a preprocessing step. The actual join does not take place during this partitioning phase. Instead, only the corresponding data partitions from both datasets are grouped together in bigger splits, referred to as *Co-Partition Splits*.

At query time, when S and T need to be joined, the join operation can take place as a map-only job, where each mapper will be assigned one complete co-partition split. As such, each mapper can join the corresponding partitions included in its split. Therefore, the join operation becomes significantly less expensive since the shuffling/sorting and reduce phases have been eliminated (compared to the traditional map-reduce join operation in Hadoop). As highlighted in Fig. 1b, the individual splits from S or T within a single co-group may have a trojan index built on them.

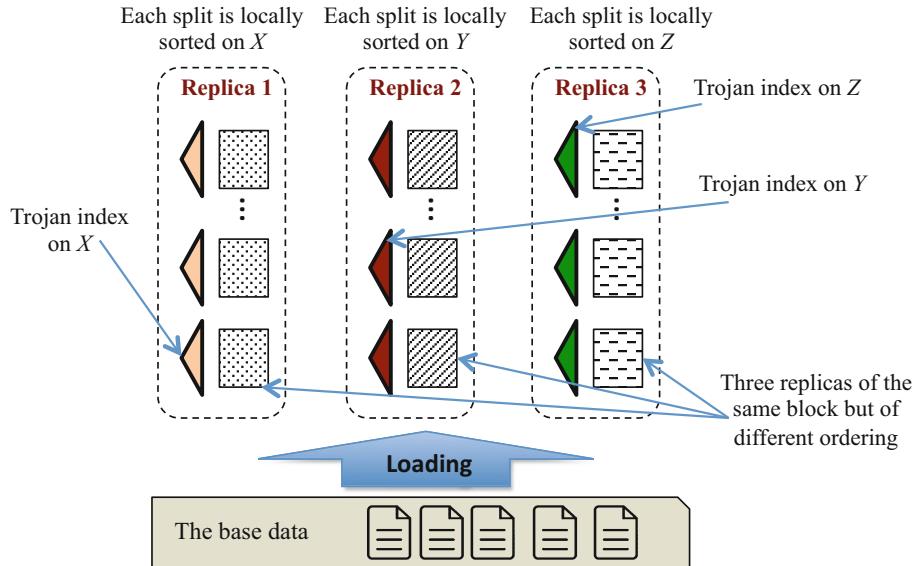
Hadoop++ framework is suitable for static indexing and joining. That is, at the time of loading the data into Hadoop, the system needs to know whether or not indexes need to be created (and on which attributes) and also whether or not co-partitioning between specific datasets needs to be performed. After loading the data, no additional indexes or co-partitioning can be created unless the entire dataset is reprocessed from scratch. Similarly, if new batches of files arrive and need to be appended to an existing indexed dataset, then the entire dataset needs to be reloaded (and

the entire indexes to be re-created) in order to accommodate for the new batches.

Record-Level Adaptive Indexing

The work proposed in Dittrich et al. (2012), Richter et al. (2012) overcomes some of the limitations of other previous indexing techniques, e.g., Dittrich et al. (2010) and Jiang et al. (2010). The key limitations include the following: First, *high creation overhead for indexes*. Usually building the indexes requires a preprocessing step, and this step can be expensive since it has to go over the entire dataset. Previous evaluations have shown that this overhead is usually redeemed from few queries, i.e., the execution of few queries using the index will redeem the cost paid upfront to create the index. Although that is true, reducing the creation overhead is always a desirable thing. The second limitation is the question of *which attributes to index?* In general, if the query workload is changing, then different indexes may need to be created (or deleted) over time. The work in Dittrich et al. (2012) and Richter et al. (2012) addresses these two limitations.

HAIL (Hadoop Aggressive Indexing Library) (Dittrich et al. 2012) makes use of the fact that Hadoop, by default, creates three replicas of each data block – this default behavior can be altered by the end users to either increase or decrease the number of replicas. In plain Hadoop, these replicas are exact mirror of each other. However, HAIL proposes to reorganize the data in each replica in a different way, e.g., each of the three replicas of the same data block can be sorted on a different attribute. As a result, a single file can have multiple clustered indexes at the same time. For example, as illustrated in Fig. 2, the 1st replica can have each of its splits sorted on attribute X , the 2nd replica sorted on attribute Y , and the 3rd replica sorted on attribute Z . These sorting orders are local within each split. Given this ordering, a clustered trojan index as proposed in Dittrich et al. (2010) can be built on each replica independently.



Big Data Indexing, Fig. 2 HAIL Indexing Framework

HAIL also proposes a *replica-aware scheduling policy*. In plain Hadoop, since all replicas are the same, the task scheduling decision does not differentiate between the replicas. In contrast in HAIL the task scheduler needs to take the query predicates into account while selecting the target replica to work on. For example, referring to Fig. 2, given a query involving a selection predicate on attribute Y , then HAIL scheduler will try to assign the map tasks to the splits of the 2nd replica. Otherwise, a full scan operation has to be performed on either of the other replicas because their indexes cannot help in evaluating the given predicate.

LIAH (Lazy Indexing and Adaptivity in Hadoop) indexing framework (Richter et al. 2012) further extends the idea of HAIL by adaptively selecting the columns to be indexed under changing workloads and also lazily building these indexes as more queries execute in the system. LIAH can incrementally build a given index starting from indexing few splits and incrementally indexing more splits as more queries are executed until the entire index is built. This strategy is based on the idea of *piggybacking* the index creation task over other user's queries to reduce the overheads involved in the index creation.

Referring to Fig. 2, in LIAH it is possible that the system starts without any indexes on the base data. And then, by automatically observing the query workload, the system decides that attributes X and Y are good candidates for indexing, e.g., many queries have selection predicates on either of these two attributes. LIAH puts a strategy to incrementally make the splits of the 1st replica sorted based on X , and for each split where its data becomes sorted, its corresponding trojan index is built. LIAH framework keeps track of which blocks have been indexed and which blocks need to be indexed (which will be done progressively and piggybacked over future users' jobs). As more user jobs are submitted to the system and the data blocks are read anyway, additional blocks can be indexed. In this way, the overheads involved in the index creation are distributed over many user queries.

Split-Level Indexing

Most of the previous indexing techniques proposed over the Hadoop infrastructure try to mimic the indexes in traditional databases in that they are record-level indexes. That is, their objective is to eliminate and skip irrelevant

records from being processed. Although these techniques show some improvements in query execution, they still encounter unnecessary high overhead during execution. For example, imagine the extreme case where a queried value x appears only in very few splits of a given file. In this case, the indexing techniques like Hadoop++ and HAIL would encounter the overheads of starting a map task for each split, reading the split headers, searching the local index associated with the split, and then reading few data records or directly terminating. These overheads are substantial in a map-reduce job and if eliminated can improve the performance.

The *E3* framework proposed in Eltabakh et al. (2013) is based on the aforementioned insight. Its objective is not to build a fine-grained record-level index but instead be more Hadoop-compliant and build a coarse-grained split-level index to eliminate entire splits whenever possible. *E3* proposes a suite of indexing mechanisms that work together to eliminate irrelevant splits to a given query before the execution, and thus map tasks start only for a potentially small subset of splits (See Fig. 3a). *E3* integrates four indexing mechanisms, which are *split-level statistics*, *inverted indexes*, *materialized views*, and *adaptive caching*; each is beneficial under specific cases. The split-level statistics are calculated for each number and date field in the dataset.

Examples of the collected statistics include the *min* and *max* values of each field in each split, and if this min-max range is very sparse, then the authors have proposed a domain segmentation algorithm to divide this range into possibly many but tighter ranges to avoid false positives (a false positive is when the index indicates that a value may exist in the dataset while it is actually not present). The inverted index is built on top of the string fields in the dataset, i.e., each string value is added to the index, and it points to all splits including this values. By combining these two types of indexes for a given query involving a selection predicate, *E3* can identify which splits are relevant to the query, and only for those splits a set of mappers will be triggered.

The other two mechanisms, i.e., materialized views and adaptive caching, are used in the cases

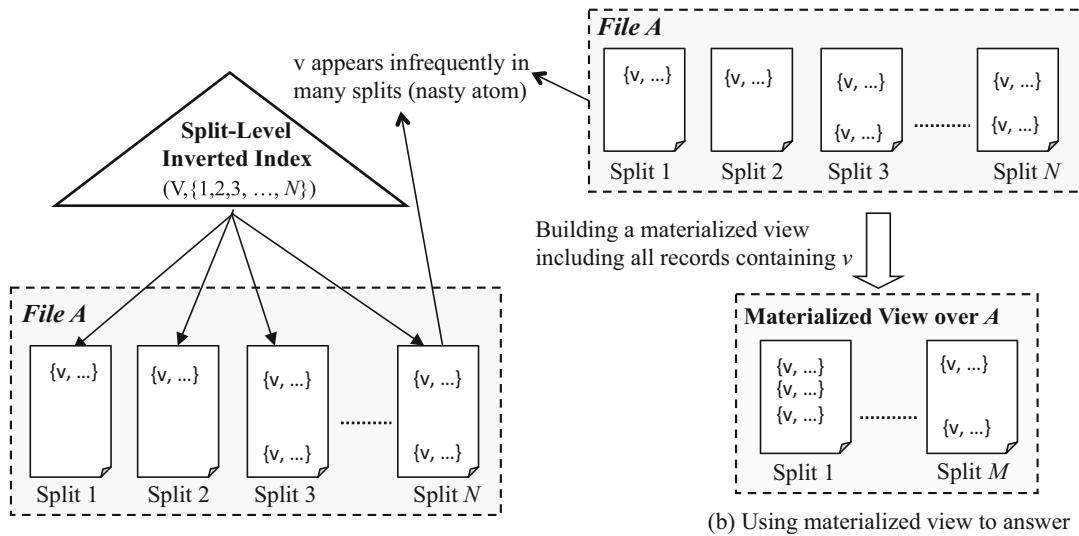
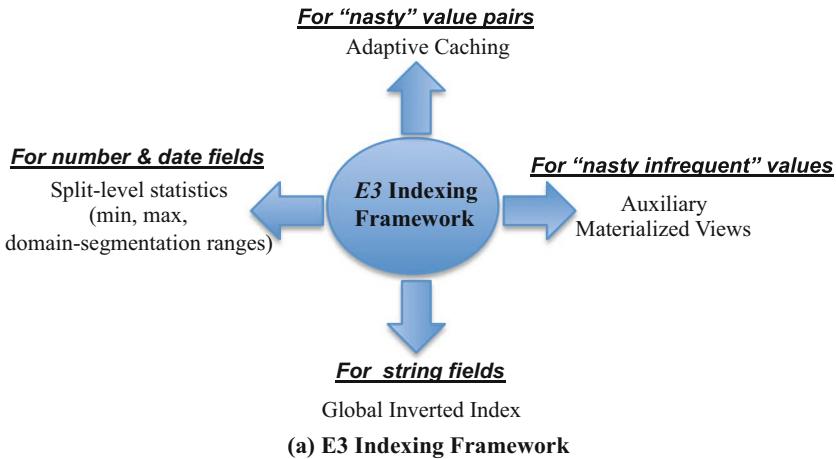
where indexes are mostly useless. One example provided in Eltabakh et al. (2013) is highlighted in Fig. 3b. This example highlights what is called “*nasty values*,” which are values that are infrequent over the entire dataset but scattered over most of the data splits, e.g., each split has one or few records of this value. In this case, the inverted index will point to all splits and becomes almost useless.

E3 framework handles these nasty values by coping their records into an auxiliary materialized view. For example, the base file *A* in Fig. 3b will now have an addition materialized view file stored in HDFS that contains a copy of all records having the nasty value v . Identifying the nasty values and deciding on which ones have a higher priority to handle have been proven to be an NP-hard problem, and the authors have proposed an approximate greedy algorithm to solve it (Eltabakh et al. 2013).

The adaptive caching mechanism in *E3* is used to optimize conjunctive predicates, e.g., ($A = x$ and $B = y$), under the cases where each of x and y individually is frequent, but their combination in one record is very infrequent. In other words, none of x or y is nasty value, but their combination is a *nasty pair*. In this case, neither the indexes nor the materialized views are useful. Since it is prohibitively expensive to enumerate all pairs and identify the nasty ones, the *E3* framework handles these nasty pairs by dynamically observing the query execution and identifying on the fly the nasty pairs. For example, for the conjunctive predicates ($A = x$ and $B = y$), *E3* consults the indexes to select a subset of splits to read. And then, it will observe the number of mappers that actually produced matching records. If the number of mappers is very small compared to the triggered ones, then *E3* identifies (x, y) to be a nasty pair. Consequently, (x, y) will be cached along with pointers to its relevant splits.

Hadoop-RDBMS Hybrid Indexing

There has been a long debate on whether or not Hadoop and database system can coexist

(b) **E3 Inverted Indexes vs. Materialized Views****Big Data Indexing, Fig. 3** E3 indexing framework (Eltabakh et al. 2013)

together in a single working environment and whether or not this strategy is beneficial. There are several successful projects that built such integration (Abouzeid et al. 2009; Gankidi et al. 2014; Floratou et al. 2014a,b; Balmin et al. 2013; Katsipoulakis et al. 2015; Tian et al. 2016).

HadoopDB is one of the early projects that brings the optimizations of relational database systems to Hadoop (Abouzeid et al. 2009). HadoopDB proposes major changes to Hadoop's

infrastructure by replacing the HDFS storage layer by a database management layer. That is, the Data Node and Task Tracker on each slave node in the Hadoop's cluster will be running an instance of a database system. This database instance replaces the HDFS layer, and thus the data on each slave node are stored and managed by the database engine.

HadoopDB will push as much of work as possible to the database engine, and as a result

all indexing capabilities and query optimizations of database systems automatically become accessible. However, the drawback of HadoopDB is that the management of dynamic scheduling and fault tolerance becomes more complicated. In addition, the integration of structured and unstructured data in the same workflow becomes tricky to perform.

Polybase (Gankidi et al. 2014) is another system that enables the integration of Hadoop and database engines. In Polybase, the HDFS datasets are defined within the database system as *external tables*. And then, users' queries can span both the data stored in the DBMS and the data stored in HDFS's external tables. At execution time, part of the query can be translated to map-reduce jobs while another part is SQL-based.

The data flow between the two systems in Polybase takes place through custom Input-Formats and Database Connectors. However, without efficient access plans to the data in the external tables, these tables can easily become a bottleneck, and the entire execution plan slows down. The work in Gankidi et al. (2014) proposes an indexing technique, called *Polybase Split-Indexing*, that creates B+-tree indexes on the HDFS datasets. These indexes reside within the database system. These indexes can be leveraged in different ways. For selection queries, they can be used as early split-level filters to identify the relevant splits in HDFS. For join queries, they can be used for performing a semi-join within the database system before retrieving HDFS's data. Moreover, they can be used as caches of *hot* HDFS data within the database system, and if a query touches only the attributes within the index, then the entire processing can be performed inside the database.

Conclusion

This entry covers various types of big data indexing techniques to speed up and enhance the data's retrieval and querying. These techniques range from record-level to split-level, nonadaptive to adaptive, and non-hybrid to hybrid. They also cover a wide range of data types includ-

ing relational-like data, spatial data, and semi-structured JSON data. The presented techniques are the backbones for numerous applications that run at scale over big data. And with the increasing complexity of these applications in terms of the size of the collected or generated data, the heterogeneity in the data types, and the growing complexity of the applied analytics and querying, the indexing techniques will play an even more important role in bringing the processing performance into the realm of feasibility.

Cross-References

- ▶ [Hadoop](#)
- ▶ [Indexing](#)
- ▶ [Inverted Index Compression](#)
- ▶ [Performance Evaluation of Big Data Analysis](#)

References

- Abadi DJ (2010) Tradeoffs between parallel database systems, Hadoop, and Hadoopdb as platforms for petabyte-scale analysis. In: SSDBM, pp 1–3
- Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A (2009) HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. In: VLDB, pp 922–933
- Abouzeid A, Bajda-Pawlikowski K, Huang J, Abadi DJ, Silberschatz A (2010) Hadoopdb in action: building real world applications. In: SIGMOD conference, pp 1111–1114
- Balmin A, Beyer KS, Ercegovac V, McPherson J, Özcan F, Pirahesh H, Shekita EJ, Sismanis Y, Tata S, Tian Y (2013) A platform for extreme analytics. IBM J Res Dev 57(3/4):4
- Bayer R, McCreight E (1972) Organization and maintenance of large ordered indexes. Acta Informatica 1(3):173–189
- Beyer K, Ercegovac V, Gemulla R, Balmin A, Eltabakh MY, Kanne CC, Özcan F, Shekita E (2011) Jaql: a scripting language for large scale semi-structured data analysis. In: PVLDB, vol 4
- Chamberlin DD, Astrahan MM, Blasgen MW, Gray JN, King WF, Lindsay BG, Lorie R, Mehl JW et al (1974) A history and evaluation of system r. In: ACM computing practices, pp 632–646
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1)
- Dittrich J, Quiané-Ruiz JA, Jindal A, Kargin Y, Setty V, Schad J (2010) Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). In: VLDB, vol 3, pp 518–529

- Dittrich J, Quiané-Ruiz J, Richter S, Schuh S, Jindal A, Schad J (2012) Only aggressive elephants are fast elephants. *PVLDB* 5(11):1591–1602
- Eldawy A, Mokbel MF (2015) Spatialhadoop: a MapReduce framework for spatial data. In: 31st IEEE international conference on data engineering (ICDE 2015), Seoul, 13–17 Apr 2015, pp 1352–1363
- Eltabakh MY, Özcan F, Sismanis Y, Haas P, Pirahesh H, Vondrak J (2013) Eagle-eyed elephant: split-oriented indexing in hadoop. In: Proceedings of the 16th international conference on extending database technology (EDBT), pp 89–100
- Floratou A, Minhas UF, Özcan F (2014a) Sql-on-Hadoop: full circle back to shared-nothing database architectures. *PVLDB* 7(12):1295–1306
- Floratou A, Özcan F, Schiefer B (2014b) Benchmarking sql-on-hadoop systems: TPC or not TPC? In: Big data benchmarking – 5th international workshop (WBDB 2014), Potsdam, 5–6 Aug 2014, pp 63–72. Revised Selected Papers
- Gankidi VR, Teletia N, Patel JM, Halverson A, DeWitt DJ (2014) Indexing HDFS data in PDW: splitting the data from the index. *PVLDB* 7(13):1520–1528
- Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data (SIGMOD'84), pp 47–57
- Jiang D, Ooi BC, Shi L, Wu S (2010) The performance of MapReduce: an in-depth study. *Proc VLDB Endow* pp 472–483
- Katsipoulakis NR, Tian Y, Ozcan F, Pirahesh H, Reinwald B (2015) A generic solution to integrate SQL and analytics for big data. In: EDBT, pp 671–676
- Liu Y, Hu S, Rabl T, Liu W, Jacobsen H, Wu K, Chen J, Li J (2014) Dgindex for smart grid: enhancing hive with a cost-effective multidimensional range index. *PVLDB* 7(13):1496–1507. <http://www.vldb.org/pvldb/vol7/p1496-liu.pdf>
- Lu P, Chen G, Ooi BC, Vo HT, Wu S (2014) Scalagist: scalable generalized search trees for MapReduce systems [innovative systems paper]. *PVLDB* 7(14):1797–1808
- Maier D (1983) Theory of relational databases. Computer Science Press, Rockville
- Moro MM, Zhang D, Tsotras VJ (2009) Hash-based Indexing. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, pp 1289–1290
- Richter S, Quiané-Ruiz J, Schuh S, Dittrich J (2012) Towards zero-overhead adaptive indexing in Hadoop. *CoRR* abs/1212.3480
- Stonebraker M, Rowe LA, Hirohama M (1990) The implementation of POSTGRES. *TKDE* 2(1):125–142
- Stonebraker M et al (2010) MapReduce and parallel DBMSs: friends or foes? *Commun ACM* 53(1):64–71. <http://doi.acm.org/10.1145/1629175.1629197>
- Tian Y, Özcan F, Zou T, Goncalves R, Pirahesh H (2016) Building a hybrid warehouse: efficient joins between data stored in HDFS and enterprise warehouse. *ACM Trans Database Syst* 41(4):21:1–21:38

Big Data Performance Characterization

► Performance Evaluation of Big Data Analysis

B

Big Data Stream Security Classification for IoT Applications

Deepak Puthal¹, Rajiv Ranjan², and Jinjun Chen³

¹Faculty of Engineering and Information Technologies, School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW, Australia

²School of Computing Science, Newcastle University, Newcastle upon Tyne, UK

³School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC, Australia

Definitions

Big Sensing Data Streams, CIA triad, Security, Threats.

Overview

This chapter provides security classification of Big Sensing Data Streams in IoT infrastructure, which focused end-to-end data communications from IoT devices to Cloud. Security classification is made based on CIA triad. Individual component of CIA triad is defined by highlighting Big Data Streams security properties. Subsequently, security properties and descriptions are explained individually. Finally, potential threats are highlighted and classified based on CIA.

Introduction

The IoT is enabled by the latest developments in RFID, smart sensors, communication technologies, and Internet protocols. The basic premise is

to have smart sensors collaborate directly without human involvement to deliver a new class of applications (Granjal et al. 2015; Puthal et al. 2016a, b). As security will be a fundamental enabling factor of most IoT applications, mechanisms must also be designed to protect communications enabled by such technologies (Sharma et al. 2017). Granjal et al. (2015) reviewed the first article on IoT communication security. Other surveys do exist that, rather than analyzing the technologies currently being designed to enable Internet communications with sensing and actuating devices, focus on the identification of security requirements and on the discussion of approaches to the design of new security mechanisms (Medaglia and Serbanati 2010; Puthal et al. 2017a; El-Sayed et al. 2017) or, on the other end, discuss the legal aspects surrounding the impact of the IoT on the security and privacy of its users (Puthal et al. 2016a, 2017b; Weber 2010).

Applications dealing with large data sets obtained via simulation or actual real-time sensor networks/social network are increasing in abundance (Puthal et al. 2015a, 2017b, Fox et al. 2004). The data obtained from real-time sources may contain certain discrepancies which arise from the dynamic nature of the source. Furthermore, certain computations may not require all the data, and hence this data must be filtered before it can be processed (Puthal et al. 2015a, 2016b, 2017c). By installing adaptive filters that can be controlled in real time, we can filter out only the relevant parts of the data, thereby improving the overall computation speed.

Nehme et al. (2009) proposed a system StreamShield system designed to address the problem of security and privacy in the data stream. They have clearly highlighted the need of two types of security in data stream, i.e., (1) the “data security punctuations” (dsps), describing the data-side security policies, and (2) the “query security punctuations” (qsps) in their paper. The advantages of such stream-centric security model include flexibility, dynamicity, and speed of enforcement. Stream processor can adapt to not only data-related but also to security-related selectivity, which helps reduce the waste

of resources, when few subjects have access to streaming data (Puthal et al. 2015b, 2017d).

- Security verification is very important in data stream in order to avoid the unwanted and corrupted data.
- Another important problem needs to address to perform the security verification in near real time.
- Security verification should not degrade the performance of stream processing engine, i.e., speed of security verification should be much more efficient than stream processing engine.

There are several applications that exist where IoT devices work as source of data stream (Puthal et al. 2017d, e). Here we list out several applications such as real-time health monitoring applications (health care), industrial monitoring, geo-social networking, home automation, war front monitoring, smart city monitoring, SCADA, event detection, disaster management, and emergency management.

From above all applications, we found data that needs to be protected from malicious attacks to maintain originality of data before it reaches at data processing center (Puthal et al. 2016b, 2017c, d; Chen et al. 2015). As the data sources are sensor nodes, it is always important to propose lightweight security solutions for data stream (Chen et al. 2015).

The rest of the paper is organized as follows: section “[Security Properties](#)” gives the security classification and properties, section “[CIA Triad](#)” highlights the CIA triad properties of big data streams, subsequently section “[Classification](#)” classifies the security issues based on CIA properties, and finally this paper concluded in section “[Conclusion](#)”.

Security Properties

The goal of security services in big data stream is to protect the information and resources from malicious attacks and misbehavior (Puthal et al. 2016a). The security requirements in big data stream include:

- **Availability:** which ensures that the data stream is accessible to authenticated user and specific applications according to the predefined features.
- **Authorization:** which ensures that only authorized user can be involved in data analysis and modification.
- **Authentication:** which ensures that the data are from the legitimate sources by providing end-to-end security services. Data should not be from any malicious sources.
- **Confidentiality:** which ensures that a given message cannot be understood by anyone other than the desired recipients.
- **Integrity:** which ensures that a message sent from sources till the data center (cloud) is not modified by malicious intermediate.
- **Nonrepudiation:** which denotes that the source cannot deny sending a message it has previously sent.
- **Freshness:** which implies that the data is recent and ensures that no adversary can replay old messages.

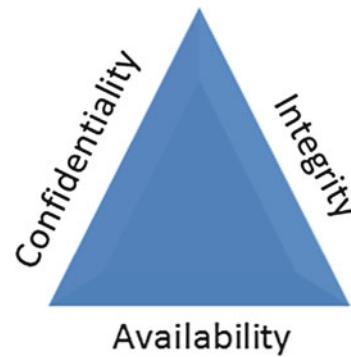
Moreover, forward and backward secrecy should also be considered, when we get the data from net set of sources.

- **Forward secrecy:** a source device should not be able to read any future messages after it leaves the network.
- **Backward secrecy:** a joining source device should not be able to read any previously transmitted message.

By considering above security requirements, we divided the security issues, threats, and solutions of IoT-generated big data stream based on CIA (confidentiality, integrity, and availability) triad. The following sections give the details about this.

CIA Triad

Confidentiality, integrity, and availability, also known as the **CIA triad**, is a model designed to guide policies for information security within



Big Data Stream Security Classification for IoT Applications, Fig. 1 CIA triad of data security either data in transit or in halt

big data stream. The **CIA triad** is shown in Fig. 1. The model is also sometimes referred to as the AIC triad (availability, integrity, and confidentiality) to avoid confusion with the Central Intelligence Agency. The elements of the triad are considered the three most crucial components of security.

Confidentiality – secrecy of the data either in transit or in halt of data. Our data stream deals with data in transit. **Def:** Confidentiality is a set of rules or a promise that limits access or places restrictions on certain types of information in data stream.

Partial confidentiality of the data is when there is considerable informational disclosure on some situation [84]. **Weak confidentiality** is in the case where some parts of the original data blocks can be reconstructed explicitly from fewer than m pieces (Bella and Paulson 1998). Information dispersal algorithms (IDAs) have weak confidentiality, and an eavesdropper can reconstruct some segments of the original file F explicitly from fewer than m pieces in the case of weak confidentiality.

Measures of the impact on confidentiality of a successful exploit of the vulnerability on the target system:

- Partial: There is considerable informational disclosure.
- Complete: A total compromise of critical system information.

Integrity – modifies the data (modify or misuse of the data) to maintain internal consistency and external consistency. **Def:** Integrity, in terms of big data stream, is the assurance that information can only be accessed or modified by those authorized users or applications. Measures taken to ensure integrity include controlling the physical environment of networked terminals and servers, restricting access to data, and maintaining rigorous authentication practices. Authentication process is the part of integrity process.

Authentication: This measures whether or not an attacker needs to be authenticated in big data stream in order to exploit the vulnerability. Authentication is required to access and exploit the vulnerability.

Measures of the impact on integrity of a successful exploit of the vulnerability on the target system:

- Partial: Considerable breach in integrity
- Complete: A total compromise of system integrity

Availability – means data must be available in the system by using controls (administrative controls, technical controls (security, firewalls, encryptions), physical controls (mass layer, i.e., access control, safety of the data)). **Def:** Data availability is a term in big data stream to ensure that data continues to be available at a required level of performance in situations ranging from normal through disastrous. In general, data availability is achieved through providing access to authenticated user and/or applications. It also works same as works at the data center for ensuring data availability (access control).

Measures of the impact on availability of a successful exploit of the vulnerability on the target system:

- Partial: Considerable lag in or interruptions in resource availability
- Complete: Total shutdown of the affected resource

Confidentiality of Big Data Streams

Confidentiality is the ability to cover messages from a passive attacker so that information in big data streams remains confidential. This is one of the most important issues for several applications such as health care, military, etc. Confidentiality must be maintained throughout the entire lifetime of the data, from source smart IoT device to long-term archiving in a cloud data center (Puthal et al. 2017b). Confidentiality is typically achieved via data encryption. This is a kind of passive attack, where unauthorized attackers monitor and listen to the communication medium. The attacks against data privacy are always passive in nature.

Integrity of Big Data Streams

Data integrity of big data streams can be defined as the overall complete consistency of the data. This can be indicated by any malicious activity, while data is transmitted between the IoT devices or between IoT and cloud data center. This can also be defined as the man-in-the-middle attack. Data integrity is generally enforced during the system setup by applying standard security rules. Data integrity can be maintained through applying validation or error checking. Message authentication code (MAC) concept is broadly applied for the data integrity in IoT infrastructure.

Availability of Big Data Streams

Data availability in big data streams is to control the access to the end users or specific applications. This subsection classifies the different access controls with data stream properties. There are two questions required for the access control process, i.e., who and what?

Classification

Table 1 presents the possible threats and attacks of IoT-generated big data stream, and the classification is based on CIA (confidentiality, integrity, and availability) triad from previous discussion. As we have classified the complete security threats and solutions over big data stream in CIA triad in previous sections, here in Table 1, we give the pictorial classifications. As seen from

Big Data Stream Security Classification for IoT Applications, Table 1 Possible threats of IoT-generated big data stream in CIA triad representation

Confidentiality	Integrity	Availability
<ul style="list-style-type: none"> Access authorization Attack against privacy Monitoring and eavesdropping Traffic analysis Camouflage adversaries 	<ul style="list-style-type: none"> Spoofed, altered, or replayed data stream information Time synchronization Selective forwarding Eavesdropping (passive attacks) Dropping data packet attacks Desynchronization Sinkhole Selfish behavior on data forwarding Sybil Wormholes Acknowledgment spoofing Hello flood attacks 	<ul style="list-style-type: none"> Mandatory access control system Role-based access control Attribute-based access control Risk-based access control View-based access control Activity-based access control Encryption-based access control Proximity-based access control Privilege state-based access control Risk-based access control Discretionary access control Information flow control model

Table 1, security threats in big data stream always fall in any properties of CIA triad.

Conclusion

The IoT infrastructure may be already visible in current deployments where networks of smart sensing devices are being interconnected with the wireless medium, and IP-based standard technologies will be fundamental in providing a common and well-accepted ground for the development and deployment of new IoT applications. According to the 4Vs' features of big data, current data stream tends to the new term as big data stream where sources are the IoT smart sensing devices. Considering that security may be an enabling factor of many of IoT applications, mechanisms to secure data stream using data in flow for the IoT will be fundamental. With such aspects in mind, in this survey we perform a basic classification of security protocols.

Cross-References

► Definition of Data Streams

References

Bella G, Paulson L (1998) Kerberos version IV: inductive analysis of the secrecy goals. In: Computer security – ESORICS 98, Louvain-la-Neuve, pp 361–375

Chen P, Wang X, Wu Y, Su J, Zhou H (2015) POSTER: iPKI: identity-based private key infrastructure for securing BGP protocol. In: 22nd ACM SIGSAC conference on computer and communications security, Colorado, pp 1632–1634

El-Sayed H, Sankar S, Prasad M, Puthal D, Gupta A, Mohanty M, Lin C (2017) Edge of things: the big picture on the integration of edge, IoT and the cloud in a distributed computing environment. IEEE Access 6:1706–1717

Fox G, Gadgil H, Pallickara S, Pierce M, Grossman R, Gu Y, Hanley D, Hong X (2004) High performance data streaming in service architecture. Technical report, Indiana University and University of Illinois at Chicago

Granjal J, Monteiro E, Sá Silva J (2015) Security for the internet of things: a survey of existing protocols and open research issues. IEEE Commun Surv Tutorials 17(3):1294–1312

Medaglia C, Serbanati A (2010) An overview of privacy and security issues in the internet of things. In: The internet of things. Springer, New York, pp 389–395

Nehme R, Lim H, Bertino E, Rundensteiner E (2009) StreamShield: a stream-centric approach towards security and privacy in data stream environments. In: ACM SIGMOD international conference on management of data, Rhode Island, pp 1027–1030

Puthal D, Nepal S, Ranjan R, Chen J (2015a) DPBSV- An efficient and secure scheme for big sensing data stream. In: 14th IEEE international conference on trust, security and privacy in computing and communications, Helsinki, pp 246–253

Puthal D, Nepal S, Ranjan R, Chen J (2015b) A dynamic key length based approach for real-time security verification of big sensing data stream. In: 16th international conference on web information system engineering, Miami, pp 93–108

Puthal D, Nepal S, Ranjan R, Chen J (2016a) Threats to networking cloud and edge datacenters in the Internet of Things. IEEE Cloud Computing 3(3):64–71

- Puthal D, Nepal S, Ranjan R, Chen J (2016b) DLSeF: A dynamic key length based efficient real-time security verification model for big data stream. *ACM Trans Embed Comput Syst* 16(2):51
- Puthal D, Mohanty S, Nanda P, Choppali U (2017a) Building security perimeters to protect network systems against cyber threats. *IEEE Consum Electron Mag* 6(4):24–27
- Puthal D, Ranjan R, Nepal S, Chen J (2017b) IoT and Big Data: an architecture with data flow and security issues. In: *Cloud infrastructures, services, and IoT systems for smart cities*, Brindisi, pp 243–252
- Puthal D, Nepal S, Ranjan R, Chen J (2017c) A dynamic prime number based efficient security mechanism for big sensing data streams. *J Comput Syst Sci* 83(1): 22–42
- Puthal D, Wu X, Nepal S, Ranjan R, Chen J (2017d, in press) SEEN: a selective encryption method to ensure confidentiality for big sensing data streams. *IEEE Trans Big Data*
- Puthal D, Obaidat M, Nanda, P, Prasad M, Mohanty S, Zomaya A (2017e, in press) Secure and sustainable load balancing of edge datacenters in fog computing. *IEEE Commun Mag*
- Sharma S, Puthal D, Tazeen S, Prasad M, Zomaya AY (2017) MSGR: a mode-switched grid-based sustainable routing protocol for wireless sensor networks. *IEEE Access* 5:19864–19875
- Weber R (2010) Internet of things—new security and privacy challenges. *Comput Law Secur Rev* 26(1):23–30

Big Data Supercomputing

► Parallel Processing with Big Data

Big Data Technologies for DNA Sequencing

Lena Wiese¹, Armin O. Schmitt², and Mehmet Gültas²

¹Institute of Computer Science, Georg-August University, Göttingen, Germany

²Department of Breeding Informatics, Georg-August University, Göttingen, Germany

Synonyms

Next-generation sequencing

Definitions

DNA sequencing is a modern technique for the precise determination of the order of nucleotides within a DNA molecule. Using this technique a huge amount of raw data is generated in life sciences.

Overview

Genome analyses play an important role in different applications in the life sciences ranging from animal breeding to personalized medicine. The technological advancements in DNA sequencing lead to vast amounts of genome data being produced and processed on a daily basis. This chapter provides an overview of the big data challenges in the area of DNA sequencing and discusses several data management solutions.

Next-generation sequencing (NGS) technologies make it possible for life scientists to produce huge amounts of DNA sequence data in a short period of time (Stephens et al. 2015). Using these technologies, in recent years thousands of genomes and short DNA sequence reads for humans, plants, animals, and microbes have been collected and explored, which enables us to develop a deeper understanding and gain new insights into the molecular mechanisms of different diseases including many types of cancer, allergies, or other disorders.

There is no doubt that NGS technologies bring considerable advantages in productivity or significant reduction in cost and time. The complexity and sheer amount of the resulting biological datasets are, however, more intricate than expected making their analysis and handling a real challenge.

Pedersen and Bongo (2016) state that off-the-shelf big data management systems might not be appropriate for an efficient and effective management of biological data: “Biological data differs in that it has more dimensions and noise, it is heterogeneous both with regards to biological content and data formats, and the statistical analysis methods are often more complex.” This

generally also applies to the specific case of DNA sequencing data. In particular, DNA data are often processed in a data analysis pipeline (like the META-pipe Pedersen and Bongo 2016) where not only the raw data but also several additional contextual metadata and provenance data are generated and have to be maintained. The raw data produced during DNA sequencing are image data generated by the sequencing hardware. Further DNA processing steps comprise:

Primary analysis: producing short DNA sequences (called reads) out of the raw data and assigning quality scores to them

Secondary analysis: assembling several short reads guided by a reference DNA sequence – this process is called read mapping – as well as analyzing the reads with respect to the reference sequence by, for example, identifying single nucleotide variants or deletions

Tertiary analysis: processing the genome data to achieve advanced analyses by integrating several data sources (like multiple DNA samples, metadata, annotations, etc.)

Big Data Challenges

Current life sciences are more and more data driven. In practice, this means that data are recorded and generated in an essentially automatic way. A large portion of life science data is taken up by DNA sequences that are, for example, used to identify the genetic basis of a disease in the medical sciences or of (wanted or unwanted) traits of cultivated plants or animals which are supposed to be used in breeding programs. It is obvious that such large bodies of data can no longer be stored and analyzed in the traditional way but can only be harnessed by advanced data management and analysis systems.

For the years around the millennium, the growth rate of computing power kept pace with the growth rate of data output of sequencing facilities, such that then data analysis was feasible even with ordinary PCs. This remarkable parallelism has ended in 2008 with the emergence of a new sequencing technology – next-generation sequencing. While computing power is predicted to follow Moore's law, that is,

roughly doubling every 24 months, the years after the advent of NGS witnessed a decline of sequencing cost that could only be measured in orders of magnitudes as can be seen in the so-called Carlson curves (Carlson 2003).

The challenges that arise by this gap between accumulation of data on the one hand and the capacities to store, analyze, and interpret them in a meaningful way are asking for new and creative, perhaps also radical ways to deal with data. We can distinguish two related, but distinct, aspects of this problem: (1) data storage and accessibility and (2) data analysis in acceptable times.

Key Research Findings

Several ways to address the big data challenges when processing genome sequencing data have been proposed in the last decade. We briefly survey several streams of research in the following subsections.

From Data Storage to Data Disposal

Thus far, the generally acknowledged, albeit in most cases, tacit agreement in the life sciences was to keep all raw data for the purpose of reproducibility or also for secondary analyses. This attitude has begun to erode in DNA sequencing projects. First, it should be clearly defined what raw data are in sequencing projects. In its most primitive form, the sequence raw data are image files which are processed in several steps to yield short sequence reads which in turn are assembled to transcripts, genes, or genomes. It has become a generally adopted practice to consider the short sequence reads, together with quality scores, as raw data which can be archived in the Sequence Read Archive (<https://www.ncbi.nlm.nih.gov/sra>). Probably unknowingly data scientists and analysts have adopted the plant breeders' wisdom: "Breeding is the art of throwing away" (Becker 2011) transforming it into "Managing data is the art of throwing away".

Sophisticated Algorithms for Data Analysis

A PubMed search has revealed a strong increase in scientific articles dealing with ultrafast algorithms suggesting that this is the response to the emergence of big data (Fig. 1). The speed of data processing in DNA sequence analysis can be increased by a combination of several factors, like storing sequence data in data structures that permit rapid access (e.g., hash tables or suffix arrays) and the development of lightweight algorithms that confine themselves to steps that are absolutely necessary. For instance, the quantification of gene expression based upon transcriptome data could be accelerated by an order of magnitude thanks to the so-called quasi-alignments. In short, it is determined via a quasi-alignment if a short sequence read and a given gene sequence match without the explicit calculation of a nucleotide-by-nucleotide alignment extending over the full length of the short read sequence. This is simply not needed for the purpose of expression quantification (Patro et al. 2017).

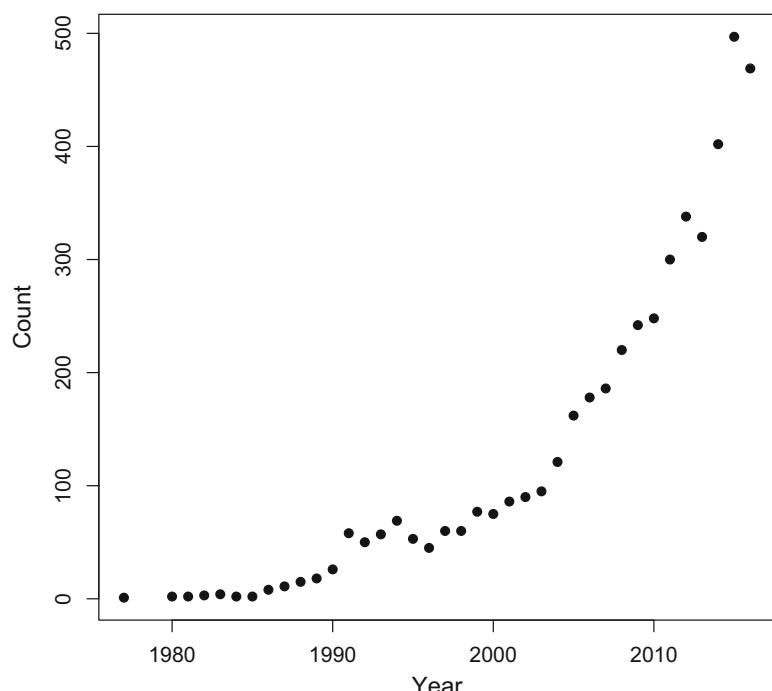
DNA-Specific Compression

Due to its string nature, specialized encodings of genomic data can significantly reduce the storage consumption – either with or without loss of accuracy. In particular, the mappings of several short DNA reads to a longer reference genome string offer benefits in terms of compressed data representation. Various lossless and lossy compression methods were devised and implemented to boil down the disk requirements. Accepting limited data losses, a compression by two orders of magnitude has turned out to be feasible (Popitsch and von Haeseler 2012). An even higher compression could be reached by the so-called delta encoding (Christley et al. 2008).

Such kinds of compression have been incorporated into standardized file formats for genome representation. The Sequence Alignment/Map (SAM) format stores the start position of a short read with respect to the reference genome, the read's actual sequence, and a CIGAR string denoting the differences between the short read sequence and the reference

Big Data Technologies for DNA Sequencing,

Fig. 1 Articles on ultrafast
algorithms in PubMed



genome, thus representing a significant reduction in disk space requirement. The lossy CRAM file format applies a heavyweight compression scheme and thus reduces the storage consumption even more (Bonfield and Mahoney 2013).

Departing from the idea of processing data stored in flat files, integrating genome-specific compression into a database system which holds data in the main memory was developed in Dorok et al. (2017). In this so-called base-centric encoding, each base is stored in a separate row of a database table such that database-specific operations can be used to analyze the genome sequence. This technique requires that the entire genome dataset fits into the main memory.

Parallel Processing and Modern Hardware Support

The accurate and fast detection of genomic variants like DNA insertions/deletions or single nucleotide polymorphisms (SNPs) based on NGS data plays an essential role in clinical researches. For this aim, several analysis pipelines combining short read aligners (e.g., BWA-MEM, Bowtie2, SOAP3, and Novoalign (<http://novocraft.com/>)) with variant callers (e.g., the Genome Analysis Toolkit HaplotypeCaller (GATK-HC), Samtools mpileup, and Torrent Variant Caller (v4.0 Life Technologies)) have been developed. However, the analysis of the raw sequence data is computationally intensive and requires significant memory consumption. In order to deal with this problem, most aligners and variant callers have been implemented in multi-threaded mode (e.g., BWA-MEM, Bowtie2, and Novoalign) or using GPU-based software (SOAP3) to ensure a feasible computation time. For details like memory usage or multithreading of the different approaches, see the review by Mielczarek and Szyda (2016); also see the references therein for details about the different tools mentioned above.

Thus, parallelization of analysis tasks is as important as the sequencing process itself and promises a huge potential for DNA processing. Several frameworks aimed at parallelization for specific applications. The MapReduce paradigm has hence received a lot of attraction. Martínez

et al. (2015) describe a distributed framework for read mapping based on the Message Passing Interface (MPI) in a cluster of nodes. They discuss the issues of splitting and distributing the inputs as well as merging the results. Already in the year 2010, an overview by Taylor (2010) surveyed applications of the Hadoop framework; more recently, other approaches based on the MapReduce framework have been developed (Chung et al. 2014). Other distributed processing systems like Spark have also received attention for applications in genome sequencing and processing (Mushtaq et al. 2017).

To benefit from advances of modern hardware technology, some DNA analysis processes have been ported to run on graphical processing units (GPUs) (Salavert Torres et al. 2012). However, these approaches incur an overhead for preprocessing the data and require highly specialized algorithms in order to take advantage of the GPU platforms.

Integration of Heterogeneous Data

Considering the growth rate of DNA sequencing data, Stephens et al. have demonstrated in their study (Stephens et al. 2015) that the sequencing technologies are one of the most important generators of big data. However, these massive datasets are often neither well structured nor organized in any consistent manner which makes their direct usage difficult. Consequently, the researchers have, first of all, to deal with the handling of big data to perform any analysis or comparison studies between different genomes. Thus, an effort for these big data challenges in life sciences is needed today to store the data in hierarchical systems in more efficient ways that make them available at different levels of analysis.

Management of biological data often requires the connection and integration of different data sources for a combined analysis. By default, biomedical text formats use identifiers (e.g., for genes or genomic variants), and different text files can only be combined by ID-based linking: data combination heavily relies on string equality matching over these IDs. To exacerbate the situation, these IDs are often hidden in larger strings such that these IDs have to be extracted first

before the data can be joined. A conjecture shared by many researchers in the field is that, on the high level of data management, explicitly linking data items in a graph structure by navigational access will enable more efficient data combination than join operations over string IDs. Hence, graph databases are deemed to be most suitable for such a data integration layer (Have and Jensen 2013) because different data sources can be combined by a link (edge) in the data graph.

One prototypical framework that integrates genome data sources and other biological datasets in a common graph-based framework is the BioGraphDB presented by Fiannaca et al. (2016) who use the OrientDB multi-model database to interconnect several text-based external data sources. The BioGraphDB system is able to process queries in the Gremlin graph query language. Textual input data is processed in this framework as follows (Fiannaca et al. 2016): “As general rule, each biological entity and its properties have been mapped respectively into a vertex and its attributes, and each relationship between two biological entities has been mapped into an edge. If a relationship has some properties, they are also saved as edge attributes. Vertices and edges are grouped into classes, according to the nature of the entities.”

Examples of Application

The raw sequencing reads are often stored in public genome repositories like:

- National Center for Biotechnology Information (<https://www.ncbi.nlm.nih.gov>),
- ENCODE (<https://www.encodeproject.org/>)
- Genome 10K Project (<https://genome10k.soe.ucsc.edu>)
- TCGA (<https://cancergenome.nih.gov>)
- Human Microbiome Project (<https://hmpdacc.org/>)

in order to make these large bodies of research data easily accessible. We survey some of these repositories with a focus on bioinformatic applications.

Bioinformatics for Sequencing Data

Bioinformatic skills play an essential role in the exploitation of the full potential of NGS data. Until now, different bioinformatic tools and algorithms have been published for the storage and computational analysis of DNA sequences. Such applications are important for the detection as well as the understanding of relevant biological processes. Currently, in the OMICtools directory (<https://omictools.com/>) for NGS data analysis, there are 47 categories with 9089 applications which are developed, for example, for data processing, quality control, genomic research, data visualization, or the identification of robust genomic associations/variants with complex diseases.

In addition to computational analysis of sequencing data, the field of bioinformatics is crucial for storage of large-scale sequencing data in databases as well as repositories. The Sequence Read Archive (SRA) (<https://www.ncbi.nlm.nih.gov/sra>) is one of the most relevant public domain repositories in which raw sequence data generated using next-generation sequencing technologies are stored for free and continuous access to the data is possible. There are further big data projects that are based on databases/repositories collecting sequencing data in life sciences:

The Encyclopedia of DNA elements (ENCODE) stores more than 15 terabytes of raw data and is thus one of the most general databases for basic biology research.

The Genome 10K Project is a collection for storage and analysis of the sequencing data corresponding to 10,000 vertebrate species. Scientists expect more than 1 petabyte by completion.

The Cancer Genome Atlas (TCGA) is an extensive effort to map the key genomic changes in 33 types of cancer. This database contains to date 2.5 petabytes of data collected from more than 11,000 patients.

Human Microbiome Project is a collection of several databases to provide quick and easy access to all publicly available microbiome data. Currently, the database contains over 14 terabytes of publicly available data in total.

Future Directions for Research

DNA sequencing is one of the major producers of big data. Novel sequencing technology will even increase the amount of DNA sequencing data produced. The introduction of mobile sequencing devices like nanopore-based technologies (Loman and Watson 2015) will turn DNA sequencing into an everyday diagnosis and monitoring tool. Though initially the available devices suffered from high error rates, ongoing technological advances will lead to an improved data quality (Jain et al. 2015).

It might be worthwhile to closely inspect genome data processing pipelines in order to identify bottlenecks. Martínez et al. (2015) report on the use of high-performance profiling that revealed idling hardware resources. By improving the task scheduling, a better runtime could be achieved. This kind of low-level performance optimization is one field of future research to improve performance of DNA processing.

Extrapolating the current drop in sequencing cost, the most radical option could be the so-called streaming algorithms for sequence analysis. With streaming algorithms, DNA sequences are analyzed in real time and are not stored at all, as shown in Cao et al. (2016).

Cross-References

- ▶ [Big Data Analysis in Bioinformatics](#)
- ▶ [GPU-Based Hardware Platforms](#)
- ▶ [Hadoop](#)

References

- Becker H (2011) Pflanzenzüchtung. UTB basics. UTB GmbH
- Bonfield JK, Mahoney MV (2013) Compression of FASTQ and SAM format sequencing data. PLoS One 8(3):e59190
- Cao MD, Ganesamoorthy D, Elliott AG, Zhang H, Cooper MA, Coin LJ (2016) Streaming algorithms for identification of pathogens and antibiotic resistance potential from real-time minion tm sequencing. GigaScience 5(1):32
- Carlson R (2003) The pace and proliferation of biological technologies. *Biosecur Bioterror Biodefense Strategy Pract Sci* 1(3):203–214
- Christley S, Lu Y, Li C, Xie X (2008) Human genomes as email attachments. *Bioinformatics* 25(2):274–275
- Chung WC, Chen CC, Ho JM, Lin CY, Hsu WL, Wang YC, Lee DT, Lai F, Huang CW, Chang YJ (2014) Clouddoe: a user-friendly tool for deploying Hadoop clouds and analyzing high-throughput sequencing data with MapReduce. *PLoS one* 9(6):e98146
- Dorok S, Breß S, Teubner J, Läpple H, Saake G, Markl V (2017) Efficiently storing and analyzing genome data in database systems. *Datenbank-Spektrum* 17(2): 139–154
- Fiannaca A, La Rosa M, La Paglia L, Messina A, Urso A (2016) Biographdb: a new graphdb collecting heterogeneous data for bioinformatics analysis. In: Proceedings of BIOTECHNO
- Have CT, Jensen LJ (2013) Are graph databases ready for bioinformatics? *Bioinformatics* 29(24):3107
- Jain M, Fiddes IT, Miga KH, Olsen HE, Paten B, Akeson M (2015) Improved data analysis for the minion nanopore sequencer. *Nat Methods* 12(4): 351–356
- Loman NJ, Watson M (2015) Successful test launch for nanopore sequencing. *Nat methods* 12(4):303
- Martínez H, Barrachina S, Castillo M, Tárraga J, Medina I, Dopazo J, Quintana-Ortí ES (2015) Scalable RNA sequencing onclusters of multicore processors. *Trustcom/BigDataSE/ISPA* 3:190–195
- Mielczarek M, Szyda J (2016) Review of alignment and SNP calling algorithms for next-generation sequencing data. *J Appl Genet* 57(1):71–79. <https://doi.org/10.1007/s13353-015-0292-7>
- Mushtaq H, Liu F, Costa C, Liu G, Hofstee P, Al-Ars Z (2017) Sparkga: a spark framework for cost effective, fast and accurate dna analysis at scale. In: Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics. ACM, pp 148–157
- Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C (2017) Salmon provides fast and bias-aware quantification of transcript expression. *Nat Methods* 14(4): 417–419
- Pedersen E, Bongo LA (2016) Big biological data management. In: Pop F, Kolodziej J, Martino BD (eds) Resource management for big data platforms. Computer communications and networks. Springer, Heidelberg, pp 265–277
- Popitsch N, von Haeseler A (2012) NGC: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic Acids Res* 41(1):e27–e27
- Salavert Torres J, Blanquer Espert I, Tomas Dominguez A, Hernendez V, Medina I, Terraga J, Dopazo J (2012) Using GPUs for the exact alignment of short-read genetic sequences by means of the burrows-wheeler transform. *IEEE/ACM Trans Comput Biol Bioinform (TCBB)* 9(4):1245–1256
- Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, Iyer R, Schatz MC, Sinha S, Robinson GE

- (2015) Big data: astronomical or genomic? PLoS Biol 13(7):e1002195
 Taylor RC (2010) An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. BMC Bioinform 11(12):S1

Big Data Visualization Tools

Nikos Bikakis
 ATHENA Research Center, Athens, Greece

Synonyms

Exploratory data analysis; Information visualization; Interactive visualization; Visual analytics; Visual exploration

Definitions

Data visualization is the presentation of data in a pictorial or graphical format, and a *data visualization tool* is the software that generates this presentation. Data visualization provides users with intuitive means to interactively explore and analyze data, enabling them to effectively identify interesting patterns, infer correlations and causalities, and supports sensemaking activities.

Overview

Exploring, visualizing, and analyzing data is a core task for data scientists and analysts in numerous applications. *Data visualization* (Throughout the article, terms *visualization* and *visual exploration*, as well as terms *tool* and *system* are used interchangeably.) (Ward et al. 2015) provides intuitive ways for the users to interactively explore and analyze data, enabling them to effectively identify interesting patterns, infer correlations and causalities, and support sensemaking activities.

The Big Data era has realized the availability of a great amount of *massive* datasets that are *dynamic*, *noisy*, and *heterogeneous* in nature.

The level of difficulty in transforming a data-curious user into someone who can access and analyze that data is even more burdensome now for a great number of users with little or no support and expertise on the data processing part. Data visualization has become a major research challenge involving several issues related to data storage, querying, indexing, visual presentation, interaction, and personalization (Bikakis and Sellis 2016; Shneiderman 2008; Godfrey et al. 2016; Idreos et al. 2015).

Given the above, modern visualization and exploration systems should effectively and efficiently handle the following aspects.

- *Real-time interaction.* Efficient and scalable techniques should support the interaction with billion object datasets while maintaining the system response in the range of a few milliseconds.
- *On-the-fly processing.* Support of on-the-fly visualizations over large and dynamic sets of volatile raw (i.e., not preprocessed) data is required.
- *Visual scalability.* Provision of effective data abstraction mechanisms is necessary for addressing problems related to visual information overloading (aka overplotting).
- *User assistance and personalization.* Encouraging user comprehension and offering customization capabilities to different user-defined exploration scenarios and preferences according to the analysis needs are important features.

The literature on visualization is extensive, covering a large range of fields and many decades. Data visualization is discussed in a great number of recent introductory-level textbooks, such as Ward et al. (2015) and Keim et al. (2010). Further, surveys of Big Data visualization systems can be found at Shneiderman (2008), Godfrey et al. (2016), Bikakis and Sellis (2016), and Idreos et al. (2015). Finally, there is a great deal of information regarding visualization tools available in the Web. We mention dataviz.tools (<http://dataviz.tools>) and [datavizcatalogue](http://datavizcatalogue.com) ([www.datavizcatalogue.com](http://datavizcatalogue.com)).

datavizcatalogue.com) which are catalogs containing a large number of visualization tools, libraries, and resources.

Visualization in Big Data Era

This section discusses the basic concepts related to Big Data visualization. First, the limitations of traditional visualization systems are outlined. Then, the basic characteristics of data visualization in the context of Big Data era are presented. Finally, the major prerequisites and challenges that should be addressed by modern exploration and visualization systems are discussed.

Traditional systems. Most *traditional exploration and visualization systems* cannot handle the size of many contemporary datasets. They restrict themselves to dealing with *small* dataset sizes, which can be easily handled and analyzed with conventional data management and visual explorations techniques. Further, they operate in an *offline* way, limited to accessing *preprocessed* sets of *static* data.

Current setting. On the other hand, nowadays, the *Big Data era* has made available large numbers of *very big* datasets that are often *dynamic* and characterized by high *variety* and *volatility*. For example, in several cases (e.g., scientific databases), new data constantly arrive (e.g., on a daily/hourly basis); in other cases, data sources offer query or API endpoints for online access and updating. Further, nowadays, an increasingly large number of *diverse users* (i.e., users with different preferences or skills) explore and analyze data in a plethora of *different scenarios*.

Modern systems. *Modern systems* should be able to efficiently handle *big dynamic* datasets, operating on machines with limited computational and memory resources (e.g., laptops). The dynamic nature of nowadays data (e.g., stream data) hinders the application of a preprocessing phase, such as traditional database loading and indexing. Hence, systems should provide *on-the-fly processing* over large sets of raw data.

Further, in conjunction with performance issues, modern systems have to address challenges related to *visual presentation*. Visualizing a large number of data objects is a challenging task; modern systems have to “*squeeze a billion records into a million pixels*” (Shneiderman 2008). Even in small datasets, offering a dataset *overview* may be extremely difficult; in both cases, *information overloading* (aka *overplotting*) is a common issue. Consequently, a basic requirement of modern systems is to effectively support *data abstraction* over enormous numbers of data objects.

Apart from the aforementioned requirements, modern systems must also satisfy the diversity of *preferences* and *requirements* posed by different *users* and *tasks*. Modern systems should provide the user with the ability to customize the exploration experience based on her preferences and the individual requirements of each examined task. Additionally, systems should automatically adjust their parameters by taking into account the *environment setting* and *available resources* (e.g., screen resolution/size, available memory).

Systems and Techniques

This section presents how state-of-the-art approaches from data management and mining, information visualization, and human-computer interaction communities attempt to handle the challenges that arise in the Big Data era.

Data reduction. In order to handle and visualize large datasets, modern systems have to deal with information overloading issues. Offering *visual scalability* is crucial in Big Data visualization. Systems should provide efficient and effective abstraction and summarization mechanisms. In this direction, a large number of systems use *approximation techniques* (aka *data reduction techniques*), in which abstract sets of data are computed. Considering the existing approaches, most of them are based on (1) *sampling* and *filtering* (Fisher et al. 2012; Park et al. 2016; Agarwal et al. 2013; Im et al. 2014; Battle et al. 2013; Bikakis et al. 2016) and/or (2) *aggregation* (e.g., binning, clustering) (Elmqvist and Fekete

2010; Bikakis et al. 2017; Jugel et al. 2016; Liu et al. 2013).

Hierarchical exploration. Approximation techniques are often defined in a hierarchical manner (Elmqvist and Fekete 2010; Bikakis et al. 2017), allowing users to explore data in different levels of detail (e.g., hierarchical aggregation).

Hierarchical approaches (sometimes also referred as *multilevel*) allow the visual exploration of very large datasets in multiple levels, offering both an overview, as well as an intuitive and effective way for finding specific parts within a dataset. Particularly, in hierarchical approaches, the user first obtains an *overview* of the dataset before proceeding to data exploration operations (e.g., roll-up, drill-down, zoom, filtering) and finally retrieving *details* about the data. Therefore, hierarchical approaches directly support the visual information seeking mantra “*overview first, zoom and filter, then details on demand.*” Hierarchical approaches can also effectively address the problem of information overloading as they adopt approximation techniques.

Hierarchical techniques have been extensively used in large *graphs visualization*, where the graph is recursively decomposed into smaller subgraphs that form a hierarchy of abstraction layers. In most cases, the hierarchy is constructed by exploiting *clustering* and *partitioning* methods (Rodrigues et al. 2013; Auber 2004; Tominski et al. 2009).

Progressive results. Data exploration requires real-time system’s response. However, computing *complete results* over large (unprocessed) datasets may be extremely costly and in several cases *unnecessary*. Modern systems should *progressively return partial* and preferably *representative* results, as soon as possible.

Progressiveness can significantly improve efficiency in *exploration scenarios*, where it is common that users attempt to find something interesting without knowing what exactly they are searching for beforehand. In this case, users perform a sequence of operations (e.g., queries), where the result of each operation determines the formulation of the next operation. In systems where progressiveness is supported, in each operation, after inspecting the already

produced results, the user is able to interrupt the execution and define the next operation, without waiting for the exact result to be computed.

In this context, several systems adopt *progressive techniques*. In these techniques the results/visual elements are computed/constructed incrementally based on user interaction or as time progresses (Stolper et al. 2014; Bikakis et al. 2017). Further, numerous recent systems integrate incremental and approximate techniques. In these cases, approximate results are computed incrementally over progressively larger samples of the data (Fisher et al. 2012; Agarwal et al. 2013; Im et al. 2014).

Incremental and adaptive processing. The dynamic setting established nowadays hinders (efficient) data preprocessing in modern systems. Additionally, it is common in exploration scenarios that only a small fragment of the input data is accessed by the user.

In situ data exploration (Alagiannis et al. 2012; Olma et al. 2017; Bikakis et al. 2017) is a recent trend, which aims at enabling on-the-fly exploration over large and dynamic sets of data, without (pre)processing (e.g., loading, indexing) the whole dataset. In these systems, *incremental* and *adaptive* processing and indexing techniques are used, in which small parts of data are processed incrementally “following” users’ interactions.

Caching and prefetching. Recall that, in exploration scenarios, a sequence of operations is performed and, in most cases, each operation is driven by the previous one. In this setting, *caching* and/or *prefetching* the sets of data that are likely to be accessed by the user in the near future can significantly reduce the response time (Battle et al. 2016; Bikakis et al. 2017). Most of these approaches use prediction techniques which exploit several factors (e.g., user behavior, user profile, use case) in order to determine the upcoming user interactions.

User assistance. The large amount of available information makes it difficult for users to manually explore and analyze data. Modern systems should provide mechanisms that assist the user and reduce the effort needed on their part,

considering the diversity of preferences and requirements posed by different users and tasks.

Recently, several approaches have been developed in the context of *visualization recommendation* (Vartak et al. 2016). These approaches recommend the most suitable visualizations in order to assist users throughout the analysis process. Usually, the recommendations take into account several factors, such as data characteristics, examined task, user preferences and behavior, etc.

Especially considering data characteristics, there are several systems that recommend the most suitable visualization technique (and parameters) based on the type, attributes, distribution, or cardinality of the input data (Key et al. 2012; Ehsan et al. 2016; Mackinlay et al. 2007; Bikakis et al. 2017). In a similar context, some systems assist users by recommending certain visualizations that reveal surprising and/or interesting data (Vartak et al. 2014; Wongsuphasawat et al. 2016). Other approaches provide visualization recommendations based on user behavior and preferences (Mutlu et al. 2016).

Examples of Applications

Visualization techniques are of great importance in a wide range of application areas in the Big Data era. The volume, velocity, heterogeneity, and complexity of available data make it extremely difficult for humans to explore and analyze data. Data visualization enables users to perform a series of analysis tasks that are not always possible with common data analysis techniques (Keim et al. 2010).

Major application domains for data visualization and analytics are *physics* and *astronomy*. Satellites and telescopes collect daily massive and dynamic streams of data. Using traditional analysis techniques, astronomers are able to identify noise, patterns, and similarities. On the other hand, visual analytics can enable astronomers to identify unexpected phenomena and perform several complex operations, which are not feasible by traditional analysis approaches.

Another application domain is *atmospheric sciences* like *meteorology* and *climatology*. In

this domain high volumes of data are collected from sensors and satellites on a daily basis. Storing these data over the years results in massive amounts of data that have to be analyzed. Visual analytics can assist scientists to perform core tasks, such as climate factor correlation analysis, event prediction, etc. Further, in this domain, visualization systems are used in several scenarios in order to capture real-time phenomena, such as hurricanes, fires, floods, and tsunamis.

In the domain of *bioinformatics*, visualization techniques are exploited in numerous tasks. For example, analyzing the large amounts of biological data produced by DNA sequencers is extremely challenging. Visual techniques can help biologist to gain insight and identify interesting “areas of genes” on which to perform their experiments.

In the Big Data era, visualization techniques are extensively used in the *business intelligence* domain. *Finance markets* are one application area, where visual analytics allow to monitor markets, identify trends, and perform predictions. Besides, *market research* is also an application area. Marketing agencies and in-house marketing departments analyze a plethora of diverse sources (e.g., finance data, customer behavior, social media). Visual techniques are exploited to realize task such as identifying trends, finding emerging market opportunities, finding influential users and communities, and optimizing operations (e.g., troubleshooting of products and services), business analysis, and development (e.g., churn rate prediction, marketing optimization).

Cross-References

- ▶ [Graph Exploration and Search](#)
- ▶ [Visualization](#)
- ▶ [Visualization Techniques](#)
- ▶ [Visualizing Semantic Data](#)

References

- Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) Blinkdb: queries with bounded errors and bounded response times on very large data. In:

- European conference on computer systems (EuroSys), Prague, 15–17 Apr 2013
- Alagiannis I, Borovica R, Branco M, Idreos S, Ailamaki A (2012) Nodb: efficient query execution on raw data files. In: ACM conference on management of data (SIGMOD), Scottsdale, pp 241–252
- Auber D (2004) Tulip - a huge graph visualization framework. In: Jünger M, Mutzel P (eds) Graph drawing software. Mathematics and visualization. Springer, Berlin/Heidelberg
- Battle L, Stonebraker M, Chang R (2013) Dynamic reduction of query result sets for interactive visualization. In: IEEE international conference on Big Data, pp 1–8
- Battle L, Chang R, Stonebraker M (2016) Dynamic prefetching of data tiles for interactive visualization. In: ACM conference on management of data (SIGMOD), San Francisco, 26 June–1 July 2016
- Bikakis N, Sellis T (2016) Exploration and visualization in the web of big linked data: a survey of the state of the art. In: Proceedings of the workshops of the EDBT/ICDT 2016 joint conference, EDBT/ICDT workshops 2016, Bordeaux, 15 Mar 2016
- Bikakis N, Liagouris J, Krommyda M, Papastefanatos G, Sellis T (2016) Graphvizdb: a scalable platform for interactive large graph visualization. In: 32nd IEEE international conference on data engineering, ICDE 2016, Helsinki, 16–20 May 2016, pp 1342–1345
- Bikakis N, Papastefanatos G, Skourla M, Sellis T (2017) A hierarchical aggregation framework for efficient multilevel visual exploration and analysis. Semant Web J 8(1):139–179
- Ehsan H, Sharaf MA, Chrysanthis PK (2016) Muve: efficient multi-objective view recommendation for visual data exploration. In: IEEE international conference on data engineering (ICDE), Helsinki, 16–20 May 2016
- Elmqvist N, Fekete J (2010) Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. IEEE Trans Vis Comput Graph 16(3):439–454
- Fisher D, Popov I, Drucker S, Schraefel MC (2012) Trust me, I'm partially right: incremental visualization lets analysts explore large datasets faster. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI '12). ACM, New York, pp 1673–1682
- Godfrey P, Gryz J, Lasek P (2016) Interactive visualization of large data sets. IEEE Trans Knowl Data Eng 28(8):2142–2157
- Idreos S, Papaemmanoil O, Chaudhuri S (2015) Overview of data exploration techniques. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data (SIGMOD '15). ACM, New York, pp 277–281
- Im J, Villegas FG, McGuffin MJ (2014) VisReduce: fast and responsive incremental information visualization of large datasets. In: 2013 IEEE international conference on Big Data (Big Data), Silicon Valley, pp 25–32
- Jugel U, Jerzak Z, Hackenbroich G, Markl V (2016) VDDa: automatic visualization-driven data aggregation in relational databases. VLDB J 53–77
- Keim DA, Kohlhammer J, Ellis GP, Mansmann F (2010) Mastering the information age - solving problems with visual analytics. In: Eurographics Association, pp 1–168, ISBN 978-3-905673-77-7
- Key A, Howe B, Perry D, Aragon C (2012) VizDeck: self-organizing dashboards for visual analytics. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data (SIGMOD '12). ACM, New York, pp 681–684
- Liu Z, Jiang B, Heer J (2013) imMens: realtime visual querying of big data. Comput Graph Forum 32(3): 421–430
- Mackinlay JD, Hanrahan P, Stolte C (2007) Show me: automatic presentation for visual analysis. IEEE Trans Vis Comput Graph 13(6):1137–1144
- Mutlu B, Veas EE, Trattner C (2016) Vizrec: recommending personalized visualizations. ACM Trans Interact Intell Syst 6(4):31:1–31:39
- Olma M, Karpathiotakis M, Alagiannis I, Athanassoulis M, Ailamaki A (2017) Slalom: coasting through raw data via adaptive partitioning and indexing. Proc VLDB Endowment 10(10):1106–1117
- Park Y, Cafarella M, Mozafari B (2016) Visualization-aware sampling for very large databases. In: 2016 IEEE 32nd international conference on data engineering (ICDE), Helsinki, pp 755–766
- Rodrigues JF Jr, Tong H, Pan J, Traina AJM, Traina C Jr, Faloutsos C (2013) Large graph analysis in the GMine system. IEEE Trans Knowl Data Eng 25(1): 106–118
- Shneiderman B (2008) Extreme visualization: squeezing a billion records into a million pixels. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD '08). ACM, New York, pp 3–12
- Stolper CD, Perer A, Gotz D (2014) Progressive visual analytics: user-driven visual exploration of in-progress analytics. IEEE Trans Vis Comput Graph 20(12): 1653–1662
- Tominski C, Abello J, Schumann H (2009) Cgv an interactive graph visualization system. Comput Graph 33(6):660–678
- Vartak M, Madden S, Parameswaran AG, Polyzotis N (2014) SEEDB: automatically generating query visualizations. Proc VLDB Endowment 7(13): 1581–1584
- Vartak M, Huang S, Siddiqui T, Madden S, Parameswaran AG (2016) Towards visualization recommendation systems. SIGMOD Rec 45(4):34–39
- Ward MO, Grinstein G, Keim D (2015) Interactive data visualization: foundations, techniques, and applications, 2nd edn. A. K. Peters, Ltd, Natick
- Wongsuphasawat K, Moritz D, Anand A, Mackinlay JD, Howe B, Heer J (2016) Voyager: exploratory analysis via faceted browsing of visualization recommendations. IEEE Trans Vis Comput Graph 22(1): 649–658

Big Data Warehouses for Smart Industries

Carlos Costa¹, Carina Andrade², and Maribel Yasmina Santos²

¹CCG – Centro de Computação Gráfica and ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

²Department of Information Systems, ALGORITMI Research Centre, University of Minho, Guimarães, Portugal

Synonyms

Industry 4.0; Smart Industry

Definition of Terms

Big Data Warehouse (BDW). A BDW can be defined as a scalable, highly performant, and flexible storage and processing system, capable of dealing with the ever-increasing volume, variety, and velocity of data, i.e., Big Data, while lowering the costs of traditional Data Warehousing architectures through the use of commodity hardware. Big Data imposes severe difficulties for traditional data storage and processing technologies, and the BDW aims to overcome these challenges and support near real-time descriptive and predictive Big Data Analytics over huge amounts of heterogeneous data (Krishnan 2013; Russom 2016; Costa et al. 2017; Santos et al. 2017).

Smart Industry. A Smart Industry can be seen as an organization chain from any industrial sector (e.g., manufacturing, services) with high digitalization levels, which supports the replication of the physical world into a virtual world, through an environment that is highly connected with the entire chain of stakeholders (e.g., suppliers, customers, partners). The Smart Industry term is directly related with the Industry 4.0 initiative, aiming to change the actual business processes and work paradigm, using several concepts like IoT, cyber-physical systems, social web, Big

Data, or robots, in order to provide intelligence to equipment and products (Kagermann et al. 2013).

B

Overview

Nowadays, we live in a world where the strong connection between people, devices, and sensors tends to produce a vast volume and variety of data, flowing at different velocities (Villars et al. 2011; Dumbill 2013). This concept is commonly defined as Big Data, which can be used to bring intelligence to several contexts, such as healthcare, cities, factories, and retail (Manyika et al. 2011; Chen et al. 2014). In the Industry 4.0 context, (Big) Data Analytics is also identified as one of the key points in this initiative, being frequently mentioned as an important way of bringing value to the organizations. This is a main requirement, due to all the data that is produced and can be used to improve businesses (Kagermann et al. 2013; Hermann et al. 2016).

At this point, the handling of this data needs more attention than the one that practitioners tend to give to traditional data. Consequently, the concept of BDW emerged as a topic of interest within Big Data systems, bringing new relevant characteristics to traditional Data Warehouses (DWs), some of which can be highlighted as (Costa and Santos 2017a):

- Flexible storage
- High scalability at low costs through the use of commodity hardware
- High performance
- Low-latency data ingestion
- Mixed and complex analytical workloads (e.g., data mining/machine learning, intensive geospatial analytics, simulations)

In an effort to guide the implementation of Big Data Warehouses (BDWs), this entry presents a flexible architecture, infrastructure, and data model that have been developed and constantly validated using performance benchmarks and examples of application in different contexts. Consequently, besides aiming to foster future

research and help practitioners to build BDWs, this entry also focuses on the relevance and value of BDWs for Smart Industries, such as the analysis of customer complaints or the identification of patterns between defective parts and production data, decreasing the time needed to fulfill complaints.

Key Research Findings

In order to design and implement a BDW, practitioners should focus on three main aspects: the architecture of the system, the technologies to use, and the data modeling technique. This section presents the foundations, i.e., models and methods (Hevner et al. 2004), to approach these same issues.

The Big Data Warehousing approach presented in this entry takes into consideration general and scarce scientific contributions regarding Big Data systems, integrating and extending them with the appropriate contributions considering the specific characteristics of BDWs previously enumerated. First, the architectural components are as aligned as possible with the Big Data Reference Architecture from the National Institute of Standards and Technology (NIST), which presents the general components of a Big Data system (NBD-PWG 2015). Second, the approach presented in this entry follows several guidelines of the Lambda Architecture (Marz and Warren 2015), motivating practitioners to store the data with the highest level of detail, storing it as a set of immutable events (if possible), and clearly dividing the workflows as batch and streaming workflows. Finally, contributions such as the Big Data Processing Flow (Krishnan 2013) and the Data Highway Concept (Kimball and Ross 2013) are also relevant in this approach, as can be seen in the following subsections.

Architecture

Logical components help practitioners in the design of a BDW. Figure 1 illustrates the BDW architecture, highlighting the several logical components and data flows.

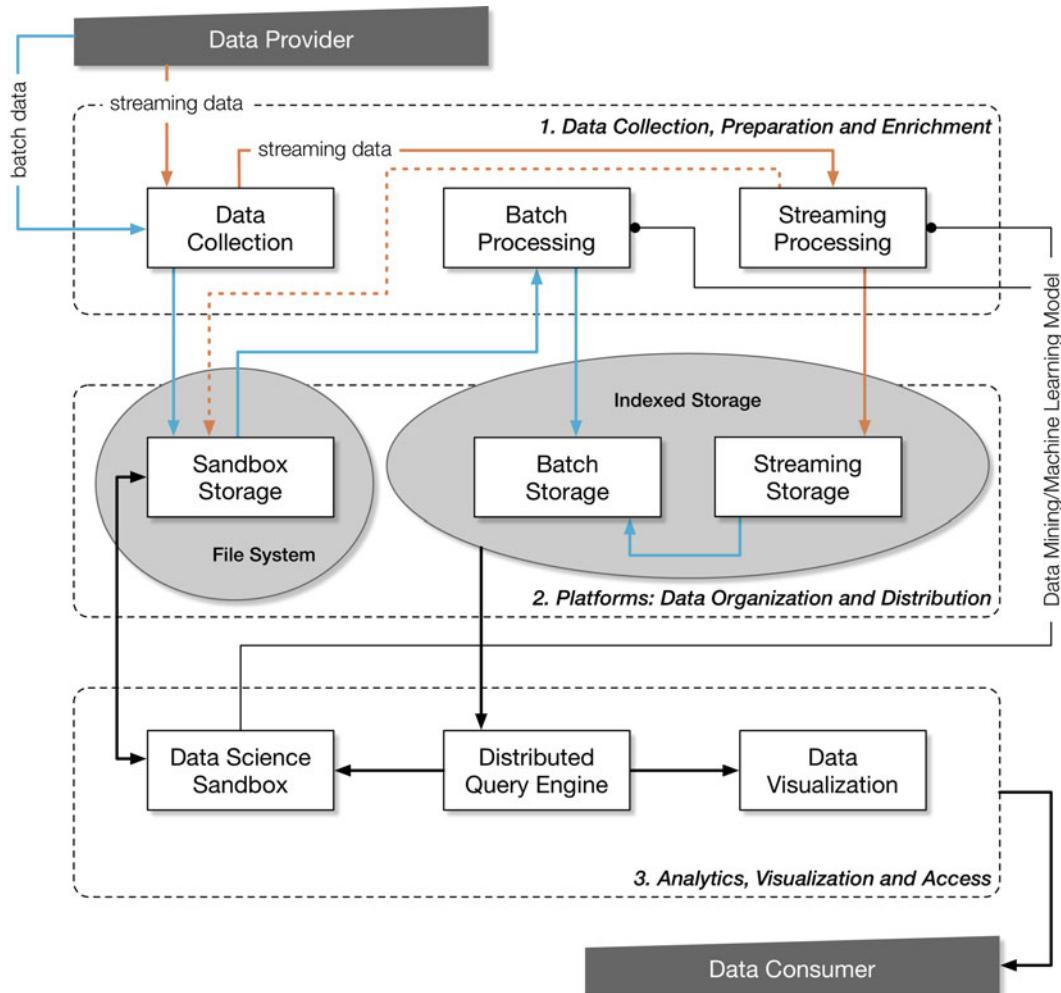
Data Collection, Preparation, and Enrichment (CPE)

Data arrives from the data provider component (e.g., users, legacy systems, Web, transactional systems, sensors), and it is collected in two distinct ways, either in batches or via streaming mechanisms. Batch data is firstly stored in the sandbox storage (distributed file system without schema restrictions) before being processed, while streaming data is immediately processed (prepared and enriched), since it typically holds more value when analyzed in near real time. Despite this, streaming data can also be stored in the sandbox storage through an optional alternative path, due to the value inherent in its raw and unprocessed state, or only as a backup strategy, for example. The Data Mining/Machine Learning models previously built by data scientists in the Data Science sandbox can be included in this stage, both in batch and streaming workloads. This works by applying the models to make predictions as data flows through the CPE workloads.

Platforms: Data Organization and Distribution

The BDW consists of three different storage areas classified into two categories: file system and indexed storage.

1. Sandbox storage – it is a highly flexible storage area without the need to rigorously define any kind of metadata. Data scientists (Costa and Santos 2017b) use the sandbox storage to “play with the data,” by identifying trends/patterns and elaborating predictions. Consequently, it is seen as a schema-less storage technology, mainly supported by highly scalable distributed file systems.
2. Batch storage – it is an indexed storage system used to store and process Big Data arriving in batches. Unlike the previous storage area, it is based on previously defined metadata to support structured analytics, although it uses a significantly more flexible data modeling technique than the typical relational DWs (see section “Data Modeling”). Its data modeling technique facilitates the integration of patterns, simulations, and predictions extracted



Big Data Warehouses for Smart Industries, Fig. 1 Big Data Warehouse architecture

and developed by data scientists. It is capable of processing huge amounts of data due to its fast sequential access capabilities.

3. Streaming storage – it is an indexed storage system that, instead of storing and processing data arriving in batches, deals with data arriving in a streaming fashion. It does not need to process the same amount of data as the batch storage, since it stores data temporarily, which is then moved to the batch storage after a certain period (e.g., 1 day, 1 week, 1 month). It can also include patterns, simulations, and predictions made by data scientists in near real time (e.g., machine learning included in

streaming processing). The streaming storage uses exactly the same data modeling technique as the batch storage, and the data stored in these two areas can be combined using an adequate distributed query engine (see Fig. 1). Ideally, for certain scenarios including update operations on data, the streaming storage technology must hold fast random access capabilities. Moreover, the streaming storage may also be supported by in-memory database technologies for fast operations on fast data, assuming that practitioners consider their level of interoperability with the chosen distributed query engine.

While the sandbox storage uses a distributed file system to act as a landing zone and as a place where data scientists can freely experiment, the batch and streaming storage areas are indexed storage systems (e.g., key-value stores, column-oriented stores, document-oriented stores) responsible for storing batch and streaming data, respectively. These two storage areas are indexed to adequately support Big Data Warehousing workloads in an effective and efficient manner. Such workloads may include ad hoc querying or self-service Business Intelligence (BI) over huge amounts of data, structured data visualization (e.g., dashboards, reports), highly intensive geospatial analytics, and visualizing simulations or predictions made by Data Mining/Machine Learning models.

Analytics, Visualization, and Access

The core component of this stage is the distributed query engine, which is able to query the data stored in the indexed storage systems, even combining batch and streaming data in a single query (see the distributed query engines in section “Technological Infrastructure”). The distributed query engine can be used to explore the data stored in the BDW but also serves the purpose of providing data to the data visualization component and to the Data Science sandbox when needed. The data consumer component represents the stakeholders interested in consuming data from the BDW (e.g., data scientists, data analysts, external users, external systems, top-level managers, operational managers).

Technological Infrastructure

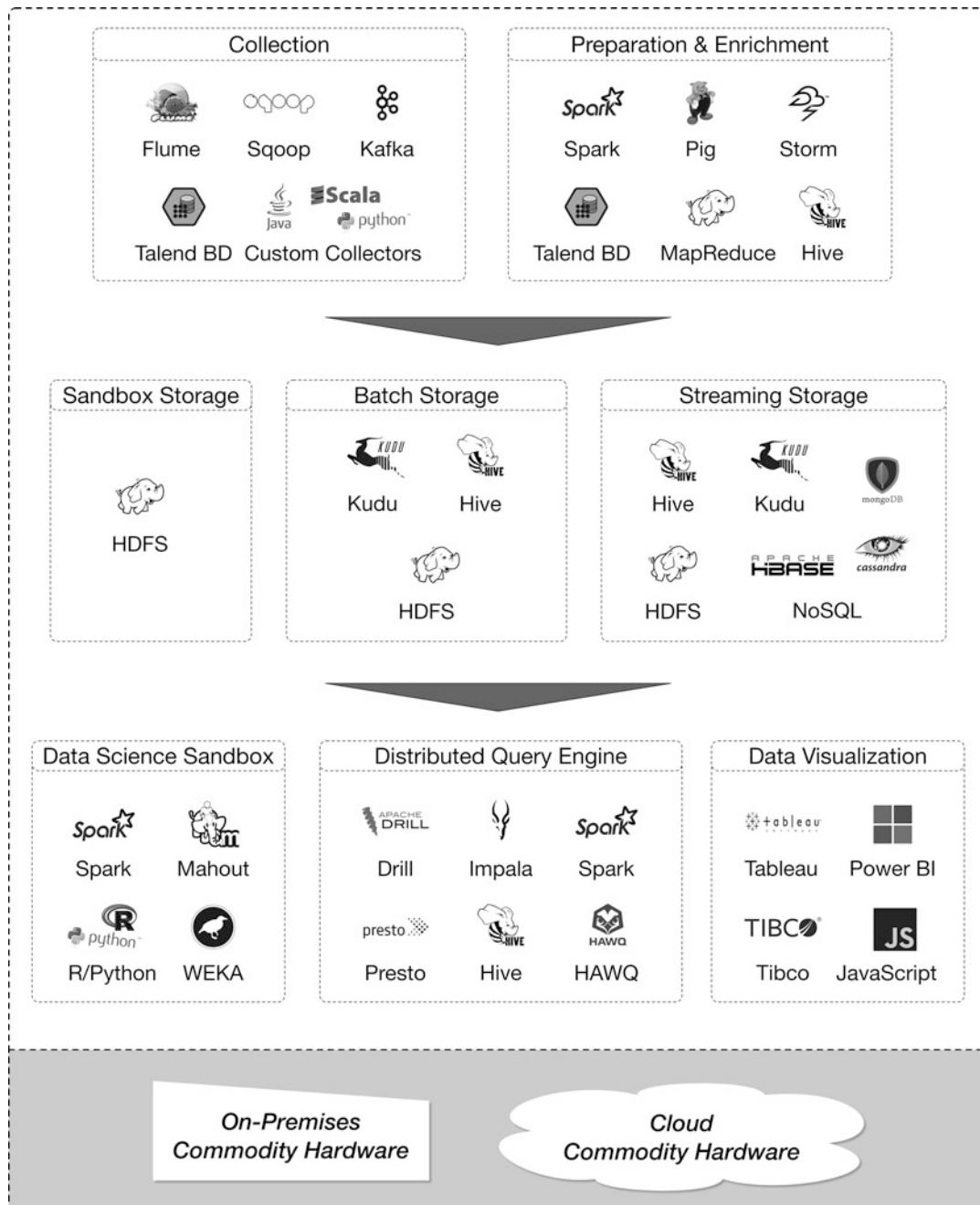
After designing the logical solution (architecture) to solve the problem, practitioners need to consider the suitable technologies to implement each logical component, in order to materialize the architecture into a physical system. Therefore, the technological infrastructure is presented in Fig. 2, which illustrates a vast set of possible technologies to implement the architecture presented above. One should be aware that there are several alternatives, as the Big Data ecosystem grows every day.

Regarding data CPE, the decision to adopt a certain technology mainly depends on the velocity of data flowing through the system. For batch data, using mechanisms such as Sqoop to collect data from relational databases, for example, custom connectors, or Talend Big Data is appropriate. However, if data flows via streaming mechanisms, it should be collected using efficient streaming systems like Kafka and Flume, for example. Once the data is collected, it is ready for preparation and enrichment. Technologies like Spark, Pig, Talend Big Data, MapReduce, and Hive are suitable for batch processing, while technologies like Spark Streaming and Storm are suitable for streaming processing. If some of these workloads include the application of previously trained Data Mining/Machine Learning models, practitioners should be aware of the level of integration between their predictive Data Science tools and the chosen technology for data preparation and enrichment.

Depending on the level of infrastructural complexity and the storage access needs (random access vs. sequential access), practitioners may consider three strategies:

1. HDFS for the sandbox storage and HDFS/Hive for the batch and streaming storage
2. HDFS for the sandbox storage, HDFS/Hive for the batch storage, and a NoSQL database for the streaming storage
3. HDFS for the sandbox storage and Kudu for the batch and streaming storage

According to laboratory experiments (Vale Lima 2017), the second strategy may offer less performance than the first one, as NoSQL databases are oriented toward online transaction processing (OLTP) applications (Cattell 2011). However, they facilitate random access operations on streaming data (e.g., updating a previously inserted record or accessing a specific record), as HDFS/Hive are sequential access-oriented systems, which typically requires recomputing entire tables/partitions to update specific records. Recently, efforts have been made to provide random access capabilities to Hive (Apache Hive 2017), but, currently, it



Big Data Warehouses for Smart Industries, Fig. 2 Technological infrastructure for Big Data Warehouses

is not a feature that can be typically seen in production environments, and it leads to several interoperability challenges (e.g., most of SQL-on-Hadoop engines lack the support to query transactional/ACID Hive tables).

Despite being faster, practitioners need to be aware that using HDFS/Hive to store streaming data accentuates the problem of having small files in Hadoop (Mackey et al. 2009). Small streaming processing intervals (micro-batches) lead to very

small files, causing severe stress on the Hadoop NameNode. Consequently, if practitioners do not want to sacrifice data timeliness, option 2 may be more suitable. If increasing streaming intervals to 30 s, 1, or 2 min, for example, does not hurt data timeliness, the HDFS files being generated will be bigger and therefore may not affect as much the performance and stability of the system. Finally, option 3 focuses on the use of technologies like Kudu for both batch and streaming data (Lipcon et al. 2015), which promises to support both scenarios without discarding the advantages of HDFS/Hive and NoSQL databases presented above. Kudu is still an emerging technology, but there is a lack of enough evidence of its performance in these scenarios.

Regarding analytical tasks, several technologies can contribute to support the Data Science sandbox, mainly in the training and testing of several Data Mining/Machine Learning models, such as Spark MLlib, Mahout, R/Python, and Weka, among many others. The distributed query engine responsible for processing data stored in the indexed storage systems of the BDW (batch and streaming storage areas) can be implemented using any SQL-on-Hadoop engine that adequately interoperates with the chosen storage technologies, including Presto, Impala, Hive, Spark SQL, Drill, and HAWQ, among others. Despite being frequently mentioned as SQL-on-Hadoop engines, these systems can process and combine data from multiple distributed systems, including NoSQL databases. Some of these engines provide interactive performance, i.e., near real-time query execution times over massive amounts of data, while others provide more robustness, scalability, and higher SQL compatibility levels. There are some scientific benchmarks available (Floratou et al. 2014; Santos et al. 2017), and practitioners are always encouraged to perform some benchmarking before adopting certain technologies for production environments. Finally, there are several data visualization technologies with appropriate connectors to these distributed query engines, such as Tableau, Power BI, and Tibco. For specific contexts, practitioners may even develop their own custom-made data

visualization platforms using JavaScript, for example.

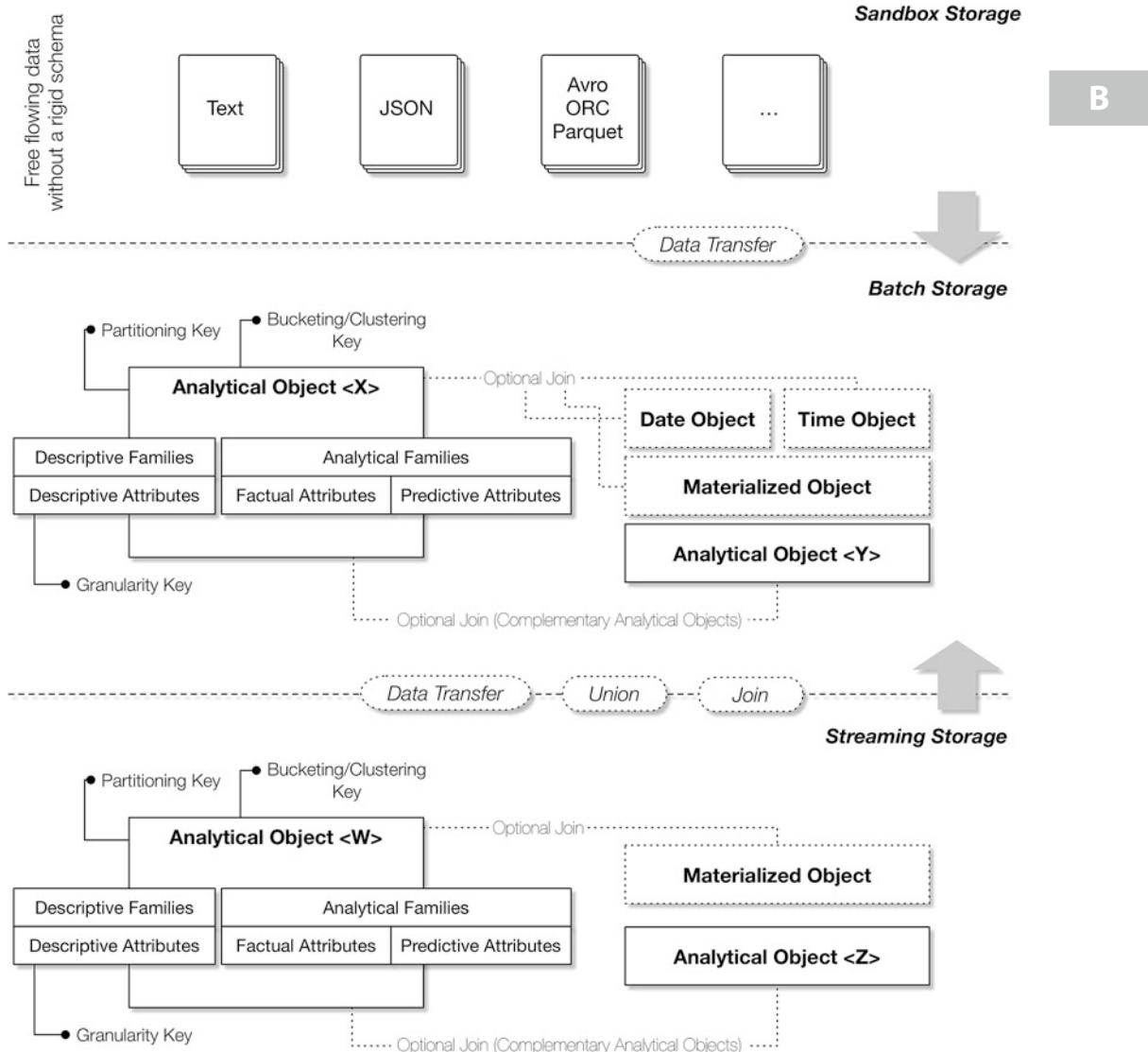
Data Modeling

The final and maybe central piece of a Big Data Warehousing system is the data modeling approach. Big Data systems are often mentioned as schema-less systems. However, for Big Data Warehousing in particular, at some point data should be structured to support the decision-making processes through adequate analytical mechanisms (e.g., data visualization of Data Mining/Machine Learning predictions, simulation results, and historical data). Having said that, this does not mean that the data model should not be flexible, quite the contrary, that is the reason why traditional ways of modeling data are not always the best solution for Big Data contexts (Costa et al. 2017), due to the volume, variety, and velocity of data sources. BDWs should have more flexible and scalable data modeling approaches, as the one depicted in Fig. 3.

Taking into consideration the description in section “Platforms: Data Organization and Distribution,” the sandbox storage can be seen as an analogy to a data lake (O’Leary 2014), where data flows in its raw form throughout a schema-less distributed file system, being stored for latter exploration and processing. Consequently, concerns with metadata and schema are not relevant at this stage. The content generated by source systems lands in this zone, despite its size or shape. As previously seen, that is mainly the case for batch data before being processed and moved to the batch storage system. In contrast, streaming data is typically processed immediately and then stored in the streaming storage, assuring latencies that satisfy the business requirements.

The batch and streaming storages use the same modeling approach, which is interpreted as follows:

1. An analytical object represents a subject of interest (e.g., sales transactions, inventory movements, social media posts, webpage clicks). It is an analogy to traditional fact tables (Kimball and Ross 2013).



Big Data Warehouses for Smart Industries, Fig. 3 Data model for Big Data Warehouses

2. Analytical objects contain descriptive and analytical families, which can be either logical or physical constructs, depending on the storage technology in use (see section “Technological Infrastructure”). These families contain a set of several attributes. Descriptive attributes are an analogy to traditional dimensions’ attributes. Factual attributes are an analogy to traditional facts in fact tables, while predictive attributes are predictions retrieved from the Data Science sandbox (e.g., sales forecasting,

customer segmentation, sentiment associated to social media post). In contrast to traditional DWs, these attributes are typically denormalized into a single analytical object (e.g., Hive table, HBase/Cassandra table) to avoid the cost of join operations on Big Data environments (Santos et al. 2017) and achieve better performance (Costa et al. 2017).

3. Despite the preference for fully denormalized structures, in certain contexts joining analyt-

ical objects may be significantly useful – see the data model in (Costa and Santos 2017a). This strategy is significantly useful when one of the analytical objects involved in the join is relatively small, it is not refreshed frequently, and/or it is reused by other analytical objects. For example, the information about customers of an organization will definitively be reused many times in different analytical objects. As mentioned above, if applicable, practitioners can create an analytical object for “customers.” This is not seen as a traditional dimension, as by itself it can contain factual and predictive attributes about customers. This strategy will save storage space (avoiding so much redundancy in Big Data environments) without significantly decreasing performance (sometimes even improving) for relatively small complementary analytical objects, as strategies like map/broadcast joins are a major selling point of several distributed query engines for Big Data (e.g., Presto).

4. For each analytical object, practitioners should define a granularity key, which may, or may not, be physically implemented through a primary key (depending on the technology being implemented), and partitioning and bucketing/clustering keys, which may significantly improve performance in certain scenarios by adequately distributing and organizing the data (Costa et al. 2017).
5. Besides analytical objects, there are three more types of optional objects. The date and time objects are responsible for storing the date and time attributes to be used by all analytical objects, which can then be used in join operations, as these objects are significantly small. Similarly to complementary analytical objects, besides standardizing information regarding date and time in the BDW, it reduces storage footprint and may even improve performance (Costa et al. 2017), as the small join operations compensate the amount of denormalized data to process. Materialized objects simply store long-running and complex queries that are frequently requested to the BDW. They can act

as a similar mechanism to traditional OLAP cubes with materialized results.

Examples of Application

This section discusses the implementation of a BDW in an organization that is working to be aligned with the Industry 4.0 initiative. In this context, a Big Data Warehousing project is being developed to help with operational and managerial decision-making. Several working areas of the factory are considered a priority in this project: customer complaints, internal defect costs, productivity associated with the several production lines, and lot release analysis. The BDW being developed follows the foundations presented in the previous sections:

- Data is collected, processed, and stored in batch and streaming storage environments.
- The technological infrastructure in use is aligned with the several technologies identified above.
- Analytical objects are adequately identified, as well as their descriptive and factual attributes. For example, Figs. 4 and 5 are data visualizations based on the “customer complaints (QCs)” analytical object, in which “customer” and “business unit” are descriptive families, and the factual information about the complaint is the analytical family.

The first dashboard (Fig. 4) aims to give a general overview about the QCs, considering several factors: the production date of the claimed parts, the business unit that produced these parts, the responsibility associated to the parts with defect (cluster), and the customers that made the complaints.

After a general analysis of the QCs, a detailed perspective on the customers that claimed some parts is presented in Fig. 5. In this dashboard, one can take conclusions regarding the relationship between the date in which the complaint was made and the production date of the claimed parts. For a simplified analysis, all complaints are colored according to the corresponding customer.

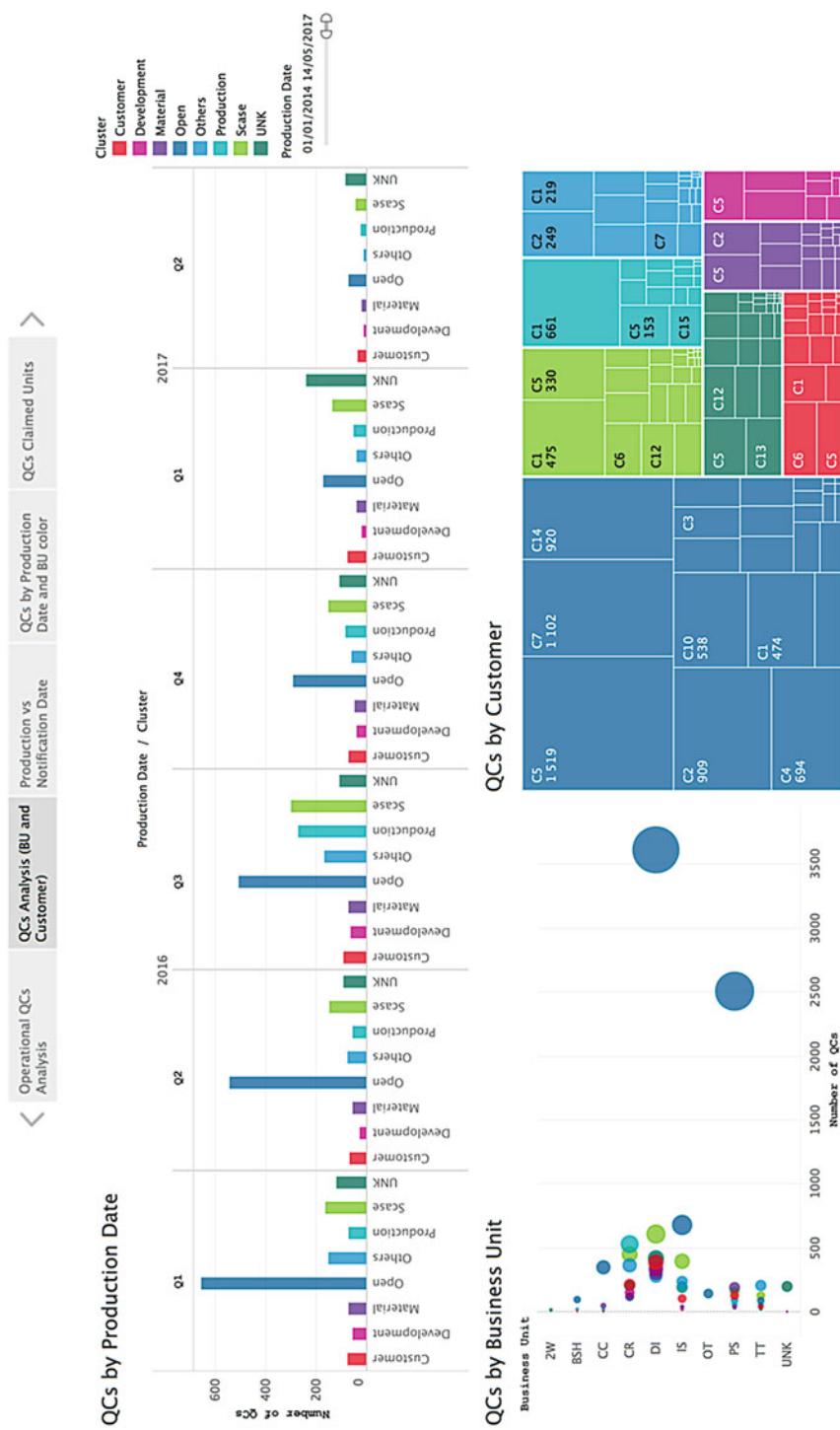


Fig. 4 General dashboard for complaints analysis (fictional data)



Big Data Warehouses for Smart Industries, Fig. 5 Complaints analysis by customer dashboard (fictional data)

Future Directions for Research

As BDWs start to be implemented in more contexts, and as technology evolves, methodological aspects may keep evolving. There are significant research contributions to be made to this topic, namely, new/extended techniques and technologies for data storage and processing. Sharing scientific benchmarks of emergent systems will also keep the community aware of different directions regarding possible architectural approaches.

Cross-References

- [Apache Hadoop](#)
- [Apache Spark](#)
- [Columnar Storage Formats](#)
- [Data Lake](#)
- [Hive](#)
- [Impala](#)
- [NoSQL Database Systems](#)
- [Spark SQL](#)
- [Storage Technologies for Big Data](#)

Acknowledgments This entry has been supported by COMPETE: POCI-01-0145-FEDER-007043 and

FCT (*Fundação para a Ciência e Tecnologia*) within the Project Scope: UID/CEC/00319/2013 and the Doctoral scholarship (PD/BDE/135101/2017) and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n° 002814; Funding Reference: POCI-01-0247-FEDER-002814].

References

- Apache Hive (2017) Apache Hive documentation. Apache Software Foundation. <https://cwiki.apache.org/confluence/display/Hive/Home>. Accessed 12 May 2017
- Cattell R (2011) Scalable SQL and NoSQL data stores. ACM SIGMOD Rec 39:12–27. <https://doi.org/10.1145/1978915.1978919>
- Chen M, Mao S, Liu Y (2014) Big data: a survey. Mob Netw Appl 19:171–209. <https://doi.org/10.1007/s11036-013-0489-0>
- Costa C, Santos MY (2017a) The SusCity Big Data Warehousing approach for smart cities. In: Proceedings of international database engineering & applications symposium, p 10
- Costa C, Santos MY (2017b) The data scientist profile and its representativeness in the European e-competence framework and the skills framework for the information age. Int J Inf Manag 37:726–734. <https://doi.org/10.1016/j.ijinfomgt.2017.07.010>

- Costa E, Costa C, Santos MY (2017) Efficient big data modelling and organization for Hadoop Hive-based data warehouses. Coimbra, Portugal
- Dumbill E (2013) Making sense of big data. *Big Data* 1: 1–2. <https://doi.org/10.1089/big.2012.1503>
- Floratou A, Minhas UF, Özcan F (2014) SQL-on-Hadoop: full circle back to shared-nothing database architectures. *Proc VLDB Endow* 7:1295–1306. <https://doi.org/10.14778/2732977.2733002>
- Hermann M, Pentek T, Otto B (2016) Design principles for Industrie 4.0 scenarios. In: 2016 49th Hawaii International Conference on System Sciences (HICSS), pp 3928–3937
- Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. *MIS Q* 28: 75–105
- Kagermann H, Wahlster W, Helbig J (2013) Recommendations for implementing the strategic initiative INDUSTRIE 4.0. National Academy of Science and Engineering, München
- Kimball R, Ross M (2013) The data warehouse toolkit: the definitive guide to dimensional modeling, 3rd edn. Wiley, Indianapolis
- Krishnan K (2013) Data warehousing in the age of big data, 1st edn. Morgan Kaufmann Publishers, San Francisco
- Lipcon T, Alves D, Burkert D, et al (2015) Kudu: storage for fast analytics on fast data. Cloudera. Unpublished paper from the KUDU team. <http://getkudu.io/kudu.pdf>
- Mackey G, Sehrish S, Wang J (2009) Improving metadata management for small files in HDFS. In: 2009 IEEE international conference on cluster computing and workshops, pp 1–4
- Manyika J, Chui M, Brown B, et al (2011) Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute
- Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning Publications Co, Shelter Island
- NBD-PWG (2015) NIST big data interoperability framework: volume 6, reference architecture. National Institute of Standards and Technology, Gaithersburg
- O'Leary DE (2014) Embedding AI and crowdsourcing in the big data lake. *IEEE Intell Syst* 29:70–73. <https://doi.org/10.1109/MIS.2014.82>
- Russom P (2016) Data warehouse modernization in the age of big data analytics. The Data Warehouse Institute, Renton
- Santos MY, Costa C, Galvão J, et al (2017) Evaluating SQL-on-Hadoop for big data warehousing on not-so-good hardware. In: Proceedings of international database engineering & applications symposium (IDEAS'17), Bristol
- Vale Lima F (2017) Big data warehousing em tempo real: Da Recolha ao Processamento de Dados. University of Minho, Guimarães
- Villars RL, Olofson CW, Eastwood M (2011) Big data: what it is and why you should care. IDC, Framingham

Big Semantic Data Processing in the Life Sciences Domain

B

Helena F. Deus

Elsevier Labs, Cambridge, MA, USA

Synonyms

Biomedical linked data; Healthcare Ontologies; Machine learning for health and life sciences

Definitions

Big semantic data processing in the life sciences deals with a set of graph-based techniques and methods used to integrate or analyze empirical evidence obtained in the course of life sciences or biomedical research.

Overview

The twofold ambition behind biomedical research and development is to either create new knowledge or apply it for treatment and prevention of disease. In life sciences, much of the research is dedicated toward understanding living systems – not just for the sake of knowledge but also to harness and control them. The promise hidden in big biomedical data processing is its potential to accelerate those efforts with predictive analytics informed by empirical results.

The mutability and adaptability inherent in all living things means that medical success requires a deep understanding of genomics: knowing how the flu virus evolves helps devise vaccines to control it, understanding which genes are controlling the proliferation of cancer means that patients can receive targeted therapies with virtually no side effects, and measuring the rate of transfer of antibiotic resistance genes in bacteria can help reduce infection spread.

The interplay between genomes and observable phenotypes is driven by a complex set of

processes. Biomedical researchers therefore tend to specialize on single processes as this facilitates the collection and analysis of empirical data. As a result, raw experimental data in biomedical settings is very heterogeneous, creating difficulties in deriving sufficiently large training sets that can be used confidently for machine learning. In fact, many biomedical experts have noticed that data in their domain is not “big” because of its volume – instead, it is “big” primarily because of its variety and velocity (Almeida et al. 2014; Ruttenberg et al. 2007; Weber et al. 2014).

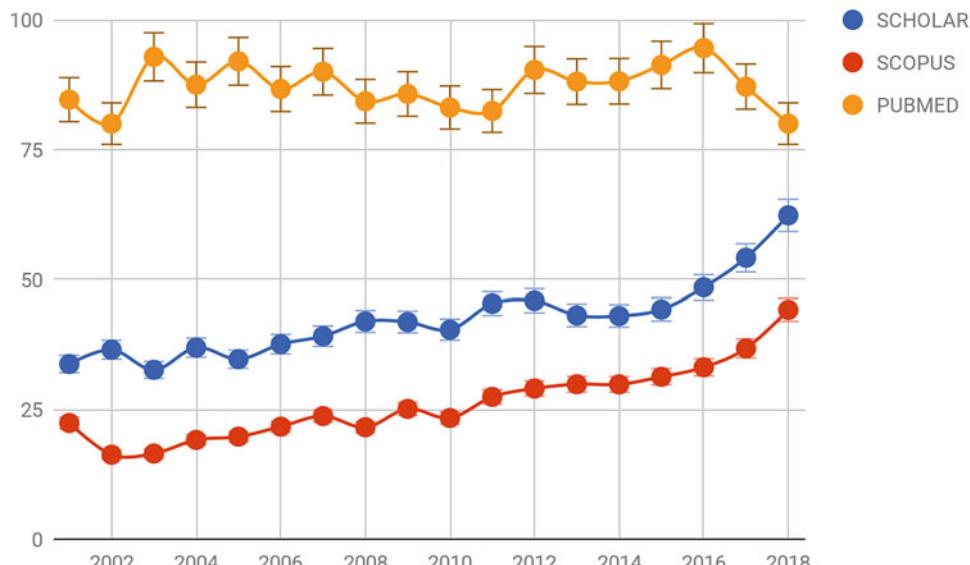
Knowledge graphs and semantic query (KG) offer a promising solution for the training set problem – if one could write a SPARQL query to recover treatment, survival time, and genetic information from cancer patients around the world, for example, it may be possible to create a training set and use it to predict which patients would likely benefit from which drugs. This chapter will present some of the KGs and research tools that have enabled and empowered biomedical scientists with such predictive biomedical analysis tools. The description in this chapter assumes a basic knowledge of semantic technologies, and whereas a knowledge of biological concepts is

not necessary, some biomedical problems will be discussed.

The KG Landscape in Biomedical Domains

The number of publications mentioning semantic web, SPARQL, knowledge graphs, or related concepts can be used as a proxy to understand where that technology sits in the “hype cycle” curve. According to google trends, the hype associated with semantic web and SPARQL peaked around 2006. However, interest in the topic has remained stable, and yearly dedicated conferences continue to attract hundreds of participants. Searches in Google Scholar, Scopus, and PubMed reveal an upward trend in the number of publications mentioning both KG and biomedical or related domains (Bio) when compared against KG across domains (Fig. 1), hinting perhaps that these technologies have achieved maturity.

KGs are also trending in industry: on January 2018 LinkedIn included the word “SPARQL” in 120 job positions and the phrase “semantic web” in 136 – many of which from very rec-



Big Semantic Data Processing in the Life Sciences Domain, Fig. 1 Percentage of papers on the topic of KG which are also in Bio domains. The decrease in PubMed for 2017 may be an artifact of incomplete indexing since the query was performed in early January 2018

ognizable biomedical industries (Eli Lilly, Amgen, Wolters Kluwer, Elsevier, Albert Einstein College of Medicine, Duke University Health System, and Clarivate). A total of 1990 Google patents results matched the word SPARQL – among which were patents from GlaxoSmithKline and Agfa HealthCare. There have also been setbacks and loss of interest in KG in Bio domains – the Conference on Semantics in Healthcare and Life Sciences (CSHALS), for example, was discontinued in 2015; the Cambridge Semantic Web Group, which was heavily influenced by the surrounding biotechnology hub, was discontinued in the summer of 2017. The latest version of the linked open data (LOD) cloud includes 332 KGs in the life sciences domain, or 28% of the whole set, but it is unclear from the Bio literature whether any of these are being used by the community. Nevertheless, the topic of KGs and SPARQL federation continues to grow and be used by a community of biomedical experts versed in these technologies. The next sections will introduce some of most successful KG in Bio domains as well as the problems they sought to address.

Gene Ontology

Founded in 1998, the gene ontology (GO) (Ashburner et al. 2000) was a visionary initiative to describe gene products across model organism databases. GO is in fact not one but three separate ontologies, all of which necessary to fully describe a gene product: biological processes, cellular components, and molecular functions. GO also includes annotations of gene products that link them to each of the three ontologies.

Usage trends: the term “gene ontology” appears 13 K times in PubMed, 156 K times in Google Scholar, and 18 K times in Scopus. It owes its popularity to the adoption by the bioinformatics community, who use it through R packages to enhance bioinformatics pipelines.

UniProt RDF

UniProt RDF is the KG exposing the UniProt protein database as a SPARQL endpoint. UniProt is the largest knowledgebase of protein sequence and annotations. Canonical

gene names are linked to associated proteins including those generated by alternative splicing (different proteins produced by the same gene), polymorphisms (variants occurring in at least 1% of the population), and/or posttranslational modifications (changes in the protein after its sequence is assembled). The key to the continued growth and popularity of UniProt has been its reliance on automated predictions that are manually curated by experts.

Usage trends: the terms “UniProt SPARQL” or variations appear 1430 times in Google Scholar, 40 times in Scopus, and 15 times in PubMed. UniProt (by itself) appears 8760 times in Scholar, 1128 times in Scopus, and 932 times in PubMed. The UniProt RDF developers have observed approximately 2000 different non-bot users querying the SPARQL endpoint, who use it as an API. The true potential of UniProt as a SPARQL endpoint has yet to be fully harnessed – however, the latest development in CRISPR–Cas9 and other gene editing technologies, together with a wish to custom design new proteins based on desired function, may become the use case that leads to an explosion in the number of uses for this endpoint.

Bio2RDF

Bio2RDF was probably the first project intentionally devised to deliver a query federation solution to the bioinformatics and systems biology communities. In 2008, when Bio2RDF was first released (Belleau et al. 2008), there were over 1000 online bioinformatics databases published annually in the NAR database issue. Frustrated by the need to create wrappers for multiple APIs, many bioinformatics organizations raised the issue of numerous obstacles to cross-database integration efforts (Haas et al. 2001; Stark et al. 2006; Fujita et al. 2010), the most significant of which was entity identifiers and attribute name alignment across sources. Bio2RDF’s main contribution was the definition of a URI creation convention which preserved the origin of the identifier or attribute while still respecting the requirement that URI is dereferenceable (<http://bio2rdf.org/<dataset>:<identifier>>, http://bio2rdf.org/<dataset>_vocabulary:<class> or

http://bio2rdf.org/<dataset>_<resource>:<identifier>_<identifier>).

Usage trends: the term “Bio2RDF” appears 1570 times in Google Scholar, 529 times in Scopus, and 20 times in PubMed. The majority of publications are in the domains of biomedical semantics or bioinformatics although some scientists working in immunology (Lamurias et al. 2017) and drug safety (Koutkias et al. 2017) have begun to explore the utility of such a valuable resource.

Linked TCGA

The Cancer Genome Atlas (TCGA) is a program initiated as a collaboration between the National Cancer Institute and the National Human Genome Research Institute and aimed at mapping genomic changes in 33 types of cancer. A total of 32K cancer cases and >3M genomic mutations were identified. The Linked TCGA project (Deus et al. 2010; Saleem et al. 2014) aimed at producing a RDF representation of the TCGA data and create links to clinical disease codes and gene identifiers. Other efforts to create RDF representation of TCGA include the project TCGA Roadmap (Robbins et al. 2013), which allows search for specific files, and the Seven Bridges Cancer Genomic Cloud also provides a SPARQL endpoint over the TCGA data with a schema very similar to Linked TCGA.

Usage trends: the terms TCGA and SPARQL appear 138 times in Google Scholar, 6 times in Scopus, and 3 times in PubMed.

Other Biomedical Knowledge Graphs

It is impossible to detail here all the KGs that have been made available to biomedical experts over the years. Some of them were never meant as more than proofs of concept and are no longer available. In other cases, those interested in recreating the work can use available RDFizers. According to the Mondeca Labs SPARQL endpoint monitor, only 37.9% of 565 SPARQL endpoints registered in datahub.io are available, and many of the unavailable ones were from biomedical domains.

In spite of setbacks, many other notable KGs are still online and more continue to be cre-

ated by consortia around the world – notably Eagle-I, a project funded by Harvard Catalyst and aimed at facilitating translational science research; the EBI RDF platform (Jupp et al. 2014), which supports a SPARQL endpoint for content in some EBI databases (BioSamples, BioModels, ChEMBL, Ensembl, Reactome, and Gene Expression Atlas), BioTea (Garcia et al. 2017), and Springer Nature SciGraph, both of which collate information about entities of significance to the research landscape. International consortia such as Allotrope, Europe’s ELIXIR initiative, Open PHACTS, and Pistoia Alliance actively seek to improve biomedical data access, interoperability, and integrity via semantic technologies.

Semantic Processing Solutions to Biomedical Problems

Some of the problems that have encouraged biomedical experts to seek KG solutions are described in this section. This is not a complete description since the field continues to evolve, yet it might provide the readers with a better understanding of this domain and how it benefits from KG processing.

Genome-Wide Association Studies (GWAS)

To understand the importance of KG processing for GWAS, it is necessary to understand how the dramatic drop in cost of genetic sequencing changed the way biomedical scientists create knowledge from empirical data. GWAS experiments are typically designed to find correlations between genes and phenotypes (e.g., cancer or diabetes). In disease cells, genes can be expressed at higher or lower concentrations than in normal cells. Statistical analysis reveals the lists of genes that may be driving the disease (i.e., gene expression signatures). GO terms associated with those lists help explain which processes are active in different components of the cell. For example, if the cell is undergoing division, cell cycle and chromosome biogenesis are more likely to come up in gene to GO mappings. Some Bio2RDF and EBI RDF datasets (e.g., Reactome) are also useful, namely, when identifying affected pathways

for drug targeting. This exploration is known as gene set enrichment analysis (GSEA). In one prominent example, GSEA was used to identify prognosis biomarkers and allow for better triaging of leukemia patients (Radich et al. 2006).

Drug Discovery and Drug Repurposing

Solutions for drug discovery and repurpose are probably two of the most well-described use cases for KGs in the Bio domain (Jentzsch et al. 2009; Chen et al. 2010; Wild et al. 2011). Before starting a clinical trial, most pharmaceutical companies first seek to understand its effect on biological entities. In many cases, the interaction between the drug and a known protein structure can be computationally predicted (i.e., in silico experiments). UniProt RDF is a key resource for this purpose, but others, including Chem2Bio2RDF and Open PHACTS, are also very relevant since they provide the necessary bridge between biological and chemical entities. Results from in silico models are often decisive in moving forward with more expensive preclinical experiments – an added incentive to aggregate extra information when building predictive models. In a recent in silico experiment (Hu et al. 2016), researchers used UniProt and the Protein Data Bank to identify five plant-based compounds and five FDA approved drugs that could target two cancer-causing genes. KGs in the chemical domain can also help biomedical experts achieve more accurate predictions by augmenting or filtering their models with chemical/biological properties such as toxicity (Kotsampasakou et al. 2017), pharmacodynamics, and absorption.

Mapping Genes to Gene Products to Phenotypes

Gene names are unique and nonoverlapping – the Human Genome Nomenclature Consortia provide scientists with a standard nomenclature. However, most protein names share a name with the gene from which they originate. Biomedical scientists adopted the convention of using italics when referring to the gene but not when referring to the protein (e.g., BRAF is the protein, and *BRAF* is the gene). This 1:1 gene/protein correspondence breaks down when

genes produce multiple proteins via alternative splicing (Black 2003). Another nomenclature issue arises when scientists uses protein family and individual protein names interchangeably (e.g., Ras is a protein family that includes H-Ras, K-Ras, and N-Ras, which are often referred to simply as “Ras proteins”). Finally, genes/proteins in different species often share a name if they share a function. MAP2K1, for example, exists in both rat (*Rattus norvegicus*) and in humans. The convention in biomedical publishing has been to use the symbol in its uppercase form (MAP2K1) when referring to the human variant but the capitalized form (Map2k1) when referring to nonhuman alternatives. This creates very obvious challenges for automated entity extraction.

The Bio2RDF KG has tackled the gene, protein, protein family, and transcript nomenclature problem with resort to a universal identifier schema that preserves the provenance of the identifiers while providing a dereferenceable description for each, including the entity type and links to related entities. With Bio2RDF, bioinformatics workflows can be simplified by resorting to a single source for identifier mapping. KGs also facilitate inference of attributes when the entities are similar – for example, knowledge that the Map2k1 protein in rat is responsible for activating cell growth can be easily inferred into its human equivalent by the use of semantic inference. At last, the problem of inconsistent phenotype and disease naming is a problem that has frustrated multiple attempts at information integration but one which has greatly benefited from KG-based solutions and is well described in Harrow et al. (2017).

Machine Learning in Bio with KGs

In the previous section, some of the most significant problems in processing of life science data were presented. The focus so far has been on data integration. This section goes a step further and explores how KGs are being used to create training sets for machine learning.

Knowledge Graphs as Aggregators of Biomedical Training Sets

The availability of graphical processing units (GPU) and cloud computing resources, coupled with easy to use machine learning libraries, is driving renewed interest in KG. Deep learning methodologies are powerful, yet they require large, clean, and standardized training sets. KGs show promise in addressing that problem. In Wang (2017), drug–drug interactions are predicted using an autoencoder network trained on a KG generated by aggregating Bio2RDF, DrugBank, and MEDLINE. In Yoshida et al. (2009), the authors describe the PosMed phenotype/candidate-gene matcher developed using curated relationships from aggregated data sources (OMIM, Reactome, MEDLINE). In Jiang et al. (2017), a “ground truth” KG of symptom–disease and examination–disease relationships from EMRs was used to identify diagnosis and misdiagnosis using machine learning. Neural networks are also being used to predict new compound–gene relationships from existing KGs (Vieira 2016).

Biomedical Ontologies and Word Embeddings

Biomedical ontologies have been used extensively to address data integration problems in the biomedical domain (Hasnain et al. 2012, 2014; Zeginis et al. 2014). Disease and phenotype ontologies are particularly helpful in grouping patients whose diagnosis may be sufficiently similar to include them in the same clinical trial cohort (Sioutos et al. 2007; Washington et al. 2009; Schriml et al. 2012). And in some cases, the optimal therapy for a patient can be inferred using a translational medicine ontology (Drolet and Lorenzi 2011). NCBO resources (Noy et al. 2009) provide significant aid to version control and reuse of existing classes and ontologies in biomedical domains. Notwithstanding, it is common for biomedical experts to work on problems for which ontologies do not exist. For example, thousands of genomic variants predictive of poor prognosis have been identified (Xu et al. 2015) but very rarely are they described in ontologies.

Word embeddings, such as Word2vec (Church 2017), can offer probabilistic approximations between keywords, which are useful and important when there is no curated ontology to resort to. Word embeddings (Yoshida et al. 2009) are being used for ontology learning from text corpora, significantly improving the rate of ontology creation and removing the burden of its creation from the domain experts (albeit curation of these automated relationships is still necessary).

Conclusions

Data in biomedical and life sciences is not “big” because of its volume – it is “big” because of its variety and heterogeneity. To engineer better therapies, study biodiversity, or prevent disease, biomedical experts need to know how changes in one dynamic system affect changes in another. Simulations are helpful, but when they are not informed and continuously updated by data, their usefulness quickly fades. Probabilistic and predictive techniques like GWAS or machine learning help automate well-defined tasks when large amounts of clean, homogeneous data are available. However, rarely are datasets large enough to address the broader – and likely more biomedically relevant – questions. The biomedical domain is rich in federated data warehouses which are diligently updated by the community. Solving the truly interesting biomedical problems may require federation of queries across them – a problem that RDF, KGs, and SPARQL technologies are well equipped to address.

Cross-References

- ▶ [Automated Reasoning](#)
- ▶ [Big Data Analysis in Bioinformatics](#)
- ▶ [Big Data for Health](#)
- ▶ [Big Data Technologies for DNA Sequencing](#)
- ▶ [Data Quality and Data Cleansing of Semantic Data](#)
- ▶ [Deep Learning on Big Data](#)
- ▶ [Distant Supervision from Knowledge Graphs](#)
- ▶ [Federated RDF Query Processing](#)

- [Ontologies for Big Data](#)
- [Semantic Interlinking](#)

References

- Almeida JS, Dress A, Kühne T, Parida L (2014) ICT for Bridging Biology and Medicine (Dagstuhl Perspectives Workshop 13342). Dagstuhl Manifestos 3:31–50. <https://doi.org/10.4230/DagMan.3.1.31>
- Ashburner M, Ball CA, Blake JA et al (2000) Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet* 25:25–29. <https://doi.org/10.1038/75556>
- Belleau F, Nolin M-A, Tourigny N et al (2008) Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *J Biomed Inform* 41:706–716. <https://doi.org/10.1016/J.JBI.2008.03.004>
- Black DL (2003) Mechanisms of alternative pre-messenger RNA splicing. *Annu Rev Biochem* 72:291–336. <https://doi.org/10.1146/annurev.biochem.72.121801.161720>
- Chen B, Ding Y, Wang H, et al (2010) Chem2Bio2RDF: A Linked Open Data Portal for Systems Chemical Biology. In: Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01. pp 232–239
- Church K (2017) Word2Vec. *Nat Lang Eng* 23:155–162. <https://doi.org/10.1017/S1351324916000334>
- Deus HF, Veiga DF, Freire PR et al (2010) Exposing The Cancer Genome Atlas as a SPARQL endpoint. *J Biomed Inform* 43:998–1008. <https://doi.org/10.1016/j.jbi.2010.09.004>
- Drolet BC, Lorenzi NM (2011) Translational research: understanding the continuum from bench to bedside. *Transl Res* 157:1–5. <https://doi.org/10.1016/j.trsl.2010.10.002>
- Fujita PA, Rhead B, Zweig AS et al (2010) The UCSC Genome Browser database: update 2011. *Nucleic Acids Res* 39:1–7. <https://doi.org/10.1093/nar/gkq963>
- Garcia A, Lopez F, Garcia L, et al (2017) Biotea, semantics for PubMed Central. <https://doi.org/10.7287/peerj.preprints.3469v1>
- Haas LM, Schwarz PM, Kodali P et al (2001) DiscoveryLink: a system for integrated access to life sciences data sources. *IBM Syst J* 40:489–511. <https://doi.org/10.1147/sj.402.0489>
- Harrow I, Jiménez-Ruiz E, Splendiani A et al (2017) Matching disease and phenotype ontologies in the ontology alignment evaluation initiative. *J Biomed Semant* 8:55. <https://doi.org/10.1186/s13326-017-0162-9>
- Hasnain A, Fox R, Decker S, Deus H (2012) Cataloguing and Linking Life Sciences LOD Cloud. In: 1st International Workshop on Ontology Engineering in a Data-driven World OEDW 2012. pp 1–11
- Hasnain A, Kamdar MR, Hasapis P, et al (2014) Linked Biomedical Dataspace: Lessons Learned Integrating Data for Drug Discovery BT. In: Mika P, Tudorache T, Bernstein A, et al. (eds) *The Semantic Web – ISWC 2014*. Springer International Publishing, Cham, pp 114–130
- Hu J-B, Dong M-J, Zhang J (2016) A holistic in silico approach to develop novel inhibitors targeting ErbB1 and ErbB2 kinases. *Trop J Pharm Res* 15:231. <https://doi.org/10.4314/tjpr.v15i2.3>
- Jentzsch A, Zhao J, Hassanzadeh O, et al (2009) Linking open drug data. In: Proc I-SEMANTICS 2009, Graz
- Jiang J, Li X, Zhao C et al (2017) Learning and inference in knowledge-based probabilistic model for medical diagnosis. *Knowl-Based Syst* 138:58–68. <https://doi.org/10.1016/J.KNOSYS.2017.09.030>
- Jupp S, Malone J, Bolleman J et al (2014) The EBI RDF platform: linked open data for the life sciences. *Bioinformatics* 30:1338–1339. <https://doi.org/10.1093/bioinformatics/btt765>
- Kotsampasakou E, Montanari F, Ecker GF (2017) Predicting drug-induced liver injury: the importance of data curation. *Toxicology* 389:139–145. <https://doi.org/10.1016/J.TOX.2017.06.003>
- Koutkias VG, Lillo-Le Louët A, Jaulent M-C (2017) Exploiting heterogeneous publicly available data sources for drug safety surveillance: computational framework and case studies. *Expert Opin Drug Saf* 16:113–124. <https://doi.org/10.1080/14740338.2017.1257604>
- Lamurias A, Ferreira JD, Clarke LA, Couto FM (2017) Generating a tolerogenic cell therapy knowledge graph from literature. *Front Immunol* 8:1–23. <https://doi.org/10.3389/fimmu.2017.01656>
- Noy NF, Shah NH, Whetzel PL et al (2009) BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Res* 37:W170–W173. <https://doi.org/10.1093/nar/gkp440>
- Radich JP, Dai H, Mao M et al (2006) Gene expression changes associated with progression and response in chronic myeloid leukemia. *Proc Natl Acad Sci U S A* 103:2794–2799. <https://doi.org/10.1073/pnas.0510423103>
- Robbins DE, Grunberg A, Deus HF et al (2013) A self-updating road map of The Cancer Genome Atlas. *Bioinformatics* 29:1333–1340. <https://doi.org/10.1093/bioinformatics/btt141>
- Ruttenberg A, Clark T, Bug W et al (2007) Advancing translational research with the Semantic Web. *BMC Bioinf* 8(Suppl 3):S2. <https://doi.org/10.1186/1471-2105-8-S3-S2>
- Saleem M, Padmanabhani SS, A-CN N et al (2014) TopFed: TCGA tailored federated query processing and linking to LOD. *J Biomed Semant* 5:47. <https://doi.org/10.1186/2041-1480-5-47>
- Schrimal LM, Arze C, Nadendla S et al (2012) Disease ontology: a backbone for disease semantic integration. *Nucleic Acids Res* 40:D940–D946. <https://doi.org/10.1093/nar/gkr972>
- Sioutos N, de Coronado S, Haber MW et al (2007) NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. *J Biomed Inform* 40:30–43. <https://doi.org/10.1016/j.jbi.2006.02.013>
- Stark C, Breitkreutz B-J, Reguly T et al (2006) BioGRID: a general repository for interaction datasets. *Nucleic*

- Acids Res 34:D535–D539. <https://doi.org/10.1093/nar/gkj109>
- Vieira A (2016) Knowledge Representation in Graphs using Convolutional Neural Networks. Comput Res Repos abs/1612.02255
- Wang M (2017) Predicting Rich Drug-Drug Interactions via Biomedical Knowledge Graphs and Text Jointly Embedding. Comput Res Repos abs/1712.08875
- Washington NL, Haendel MA, Mungall CJ et al (2009) Linking human diseases to animal models using ontology-based phenotype annotation. PLoS Biol 7:e1000247. <https://doi.org/10.1371/journal.pbio.1000247>
- Weber GM, Mandl KD, Kohane IS (2014) Finding the missing link for big biomedical data. JAMA 311:2479–2480. <https://doi.org/10.1001/jama.2014.4228>
- Wild DJ, Ding Y, Sheth AP et al (2011) Systems chemical biology and the Semantic Web: what they mean for the future of drug discovery research. Drug Discov Today. <https://doi.org/10.1016/j.drudis.2011.12.019>
- Xu N, Li Y, Zhou X et al (2015) CDKN2 gene deletion as poor prognosis predictor involved in the progression of adult B-lineage acute lymphoblastic leukemia patients. J Cancer 6:1114–1120. <https://doi.org/10.7150/jca.11959>
- Yoshida Y, Makita Y, Heida N et al (2009) PosMed (Positional Medline): prioritizing genes with an artificial neural network comprising medical documents to accelerate positional cloning. Nucleic Acids Res 37:W147–W152. <https://doi.org/10.1093/nar/gkp384>
- Zeginis D, Hasnain A, Loutas N et al (2014) A collaborative methodology for developing a semantic model for interlinking cancer chemoprevention linked-data sources. Semant Web 5(2):127–142. <https://doi.org/10.3233/SW-130112>

Big Semantic Data Processing in the Materials Design Domain

Patrick Lambrix¹, Rickard Armiento¹, Anna Delin², and Huanyu Li¹

¹Linköping University, Swedish e-Science Research Centre, Linköping, Sweden

²Royal Institute of Technology, Swedish e-Science Research Centre, Stockholm, Sweden

Definitions

To speed up the progress in the field of materials design, a number of challenges related to big data

need to be addressed. This entry discusses these challenges and shows the semantic technologies that alleviate the problems related to variety, variability, and veracity.

Overview

Materials design and materials informatics are central for technological progress, not the least in the green engineering domain. Many traditional materials contain toxic or critical raw materials, whose use should be avoided or eliminated. Also, there is an urgent need to develop new environmentally friendly energy technology. Presently, relevant examples of materials design challenges include energy storage, solar cells, thermoelectrics, and magnetic transport (Ceder and Persson 2013; Jain et al. 2013; Curtarolo et al. 2013).

The space of potentially useful materials yet to be discovered – the so-called *chemical white space* – is immense. The possible combinations of, say, up to six different elements constitute many billions. The space is further extended by possibilities of different phases, low-dimensional systems, nanostructuring, and so forth, which adds several orders of magnitude. This space was traditionally explored by experimental techniques, i.e., materials synthesis and subsequent experimental characterization. Parsing and searching the full space of possibilities this way are however hardly practical. Recent advances in condensed matter theory and materials modeling make it possible to generate reliable materials data by means of computer simulations based on quantum mechanics (Lejaeghere et al. 2016). High-throughput simulations combined with machine learning can speed up progress significantly and also help to break out of local optima in composition space to reveal unexpected solutions and new chemistries (Gaultois et al. 2016).

This development has led to a global effort – known as the Materials Genome Initiative (<https://www.mgi.gov/>) – to assemble and curate databases that combine experimentally known and computationally predicted materials

properties. A central idea is that materials design challenges can be addressed by searching these databases for entries with desired combinations of properties. Nevertheless, these data sources also open up for *materials informatics*, i.e., the use of big data methodology and data mining techniques to discover new physics from the data itself. A workflow for such a discovery process can be based on a typical data mining process, where key factors are identified, reduced, and extracted from heterogeneous databases, similar materials are identified by modeling and relationship mining, and properties are predicted through evaluation and understanding of the results from the data mining techniques (Agrawal and Alok 2016). The use of the data in such a workflow requires addressing problems with data integration, provenance, and semantics, which remains an active field of research.

Even when a new material has been invented and synthesized in a lab, much work remains before it can be deployed. Production methods allowing manufacturing the material at large scale in a cost-effective manner need to be developed, and integration of the material into the production must be realized. Furthermore, life-cycle aspects of the material need to be assessed. Today, this post-invention process takes typically about two decades (Mulholland and Paradiso 2016; Jain et al. 2013). Shortening this time is in itself an important strategic goal, which could be realized with the help of an integrated informatics approach (Jain et al. 2013, Materials Genome Initiative <https://www.mgi.gov/>).

To summarize, it is clear that materials data, experimental as well as simulated, has the potential to speed up progress significantly in many steps in the chain starting with materials discovery, all the way to marketable product. However, the data needs to be suitably organized and easily accessible, which in practice is highly nontrivial to achieve. It will require a multidisciplinary effort and the various conventions and norms in use need to be integrated. Materials data is highly heterogeneous and much of it is currently hidden behind corporate walls (Mulholland and Paradiso 2016).

Big Data Challenges

To implement the data-driven materials design workflow, we need to deal with several of the big data properties (e.g., Rajan 2015).

Volume refers to the quantity of the generated and stored data. The size of the data determines the value and potential insight. Although the experimental materials science does not generate huge amounts of data, computer simulations with accuracy comparable to experiments can. Moreover, going from state-of-the-art static simulations at temperature $T=0\text{ K}$ toward realistic descriptions of materials properties at temperatures of operation in devices and tools will raise these amounts as well.

Variety refers to the type and nature of the data. The materials databases are heterogeneous in different ways. They store different kinds of data and in different formats. Some databases contain information about materials crystal structure, some about their thermochemistry, and others about mechanical properties. Moreover, different properties may have the same names, while the same information may be represented differently in different databases.

Velocity refers to the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. In computational materials science, new data is generated continuously, by a large number of groups all over the world. In principle, one can store summary results and data streams from a specific run as long as one needs (days, weeks, years) and analyze it afterward. However, to store all the data indefinitely may be a challenge. Some data needs to be removed as the storage capacity is limited.

Variability deals with the consistency of the data. Inconsistency of the data set can hamper processes to handle and manage it. This can occur for single databases as well as data that was integrated from different sources.

Veracity deals with the quality of the data. This can vary greatly, affecting accurate analysis. The data generated within materials science may contain errors, and it is often noisy. The quality of the data is different in different databases. It may be

challenging to have provenance information from which one can derive the data quality. Not all the computed data is confirmed by lab experiments. Some data is generated by machine learning and data mining algorithms.

Sources of Data and Semantic Technologies

Although the majority of materials data that has been produced by measurement or through predictive computation have not yet become organized in general easy-to-use databases, several sizable databases and repositories do exist. However, as they are heterogeneous in nature, semantic technologies are important for the selection and integration of the data to be used in the materials design workflow. This is particularly important to deal with variety, variability, and veracity.

Within this field the use of semantic technologies is in its infancy with the development of ontologies and standards. Ontologies aim to define the basic terms and relations of a domain of interest, as well as the rules for combining these terms and relations. They standardize terminology in a domain and are a basis for semantically enriching data, integration of data from different databases (variety), and reasoning over the data (variability and veracity). According to Zhang et al. (2015a) in the materials domain ontologies have been used to organize materials knowledge in a formal language, as a global conceptualization for materials information integration (e.g., Cheng et al. 2014), for linked materials data publishing, for inference support for discovering new materials, and for semantic query support (e.g., Zhang et al. 2015b, 2017).

Further, standards for exporting data from databases and between tools are being developed. These standards provide a way to exchange data between databases and tools, even if the internal representations of the data in the databases and tools are different. They are a prerequisite for efficient materials data infrastructures that allow for the discovery of new materials (Austin 2016). In several cases the standards formalize the

description of materials knowledge (and thereby create ontological knowledge).

In the remainder of this section, a brief overview of databases, ontologies, and standards in the field is given.

Databases

The Inorganic Crystal Structure Database (ICSD, <https://icsd.fiz-karlsruhe.de/>) is a frequently utilized database for completely identified inorganic crystal structures, with nearly 200 k entries (Belsky et al. 2002; Bergerhoff et al. 1983). The data contained in ICSD serve as an important starting point in many electronic structure calculations. Several other crystallographic information resources are also available (Glasser 2016). A popular open-access resource is the Crystallography Open Database (COD, <http://www.crystallography.net/cod/>) with nearly 400 k entries (Grazulis et al. 2012).

At the International Centre for Diffraction Data (ICDD, <http://www.icdd.com/>) a number of databases for phase identification are hosted. These databases have been in use by experimentalists for a long time.

Springer Materials (<http://materials.springer.com/>) contains among many other data sources, the well-known Landolt-Börnstein database, an extensive data collection from many areas of physical sciences and engineering. Similarly, The Japan National Institute of Material Science (NIMS) Materials Database MatNavi (http://mits.nims.go.jp/index_en.html) contains a wide collection of mostly experimental but also some computational electronic structure data.

Thermodynamical data, necessary for computing phase diagrams with the CALPHAD method, exist in many different databases (Campbell et al. 2014). Open-access databases with relevant data can be found through OpenCalphad (<http://www.opencalphad.com/databases.html>).

Databases of results from electron structure calculations have existed in some form for several decades. In 1978, Moruzzi, Janak, and Williams published a book with computed electronic properties such as, e.g., density of states, bulk modulus, and cohesive energy of all metals (Moruzzi et al. 2013). Only recently, however, the use of

such databases has become widespread, and some of these databases have grown to a substantial size.

Among the more recent efforts to collect materials properties obtained from electronic structure calculations publicly available, a few prominent examples include the Electronic Structure Project (ESP) (<http://materialsgenome.se>) with ca 60k electronic structure results, Aflow (Curtarolo et al. 2012, <http://aflowlib.org/>) with data on over 1.7 million compounds, the Materials Project with data on nearly 70k inorganic compounds (Jain et al. 2013, <https://materialsproject.org/>), the Open Quantum Materials Database (OQMD, <http://oqmd.org/>), with over 470k entries (Saal et al. 2013), and the NOMAD repository with 44 million electronic structure calculations (<https://repository.nomad-coe.eu/>). Also available is the Predicted Crystallography Open Database (PCOD, <http://www.crystallography.net/pcod/>) with over 1 million predicted crystal structures, which is a project closely related to COD.

As the amount of computed data grows, the need for informatics infrastructure also increases. Many of the databases discussed above have made their frameworks available, and well-known examples include the ones by Materials Project and OQMD. Other publicly available frameworks used in publications for materials design and informatics include the Automated Interactive Infrastructure and Database for Computational Science (AiiDA, <http://www.aiida.net/>) (Pizzi et al. 2016), the Atomic Simulation Environment (ASE, <https://wiki.fysik.dtu.dk/ase/>) (Larsen et al. 2017), and the high-throughput toolkit (httk, <http://www.httk.org>) (Faber et al. 2016).

Ontologies

We introduce the features of current materials ontologies from materials (Table 1) and a knowledge representation perspective (Table 2), respectively.

Most ontologies focus on specific sub-domains of the materials field (Domain in Table 1) and have been developed with a specific use in mind (Application Scenario in Table 1).

The Materials Ontology in Ashino (2010) was designed for data exchange among thermal property databases. Other ontologies were built to enable knowledge-guided materials design or new materials discovery, such as PREMAP ontology (Bhat et al. 2013) for steel mill products, MatOnto ontology (Cheung et al. 2008) for oxygen ion-conducting materials in the fuel cell domain, and SLACKS ontology (Premkumar et al. 2014) that integrates relevant product life-cycle domains which consist of engineering analysis and design, materials selection, and manufacturing. The FreeClassOWL ontology (Radinger et al. 2013) is designed for the construction and building materials domain and supports semantic search for construction materials. MMOY ontology (Zhang et al. 2016) captures metal materials knowledge from Yago. The ontology design pattern in Vardeman et al. (2017) models and allows for reasoning about material transformations in the carbon dioxide and sodium acetate productions by combining baking soda and vinegar. Some ontologies are generated (Data source in Table 1) by extracting knowledge from other data resources such as the Plinius ontology (van der Vet et al. 1994) which is extracted from 300 publication abstracts in the domain of ceramic materials, and MatOWL (Zhang et al. 2009) which is extracted from MatML schema data to enable ontology-based data access. The ontologies may also use other ontologies as a basis, for instance, MatOnto that uses DOLCE (Gangemi et al. 2002) and EXPO (Soldatova and King 2006).

From the knowledge representation perspective (Table 2), the basic terms defined in materials ontologies involve materials, properties, performance, and processing in specific sub-domains. The number of concepts ranges from a few to several thousands. There are relatively few relationships and most ontologies have instances. Almost all ontologies use OWL as a representation language. In terms of organization of materials ontologies, Ashino's Materials Ontology, MatOnto, and PREMAP ontology are developed as several ontology components that are integrated in one ontology. In Table 2 this is denoted in the modularity column.

Big Semantic Data Processing in the Materials Design Domain, Table 1 Comparison of materials ontologies from a materials perspective

Materials ontology	Data source	Domain	Application scenario
Ashino's Materials Ontology (Ashino 2010)	Thermal property databases	Thermal properties	Data exchange, search
Plinius ontology (van der Vet et al. 1994)	Publication abstracts	Ceramics	Knowledge extraction
MatOnto (Cheung et al. 2008)	DOLCE ontology ^a , EXPO ontology ^b	Crystals	New materials discovery
PREMAP ontology (Bhat et al. 2013)	PREMAP platform	Materials	Knowledge-guided design
FreeClassOWL (Radinger et al. 2013)	Eurobau data ^c , GoodRelations ontology ^d	Construction and building materials	Semantic query support
MatOWL (Zhang et al. 2009)	MatML schema data	Materials	Semantic query support
MMOY (Zhang et al. 2016)	Yago data	Metals	Knowledge extraction
ELSSI-EMD ontology (CEN 2010)	Materials testing data from ISO standards	Materials testing, ambient temperature tensile testing	Data interoperability
SLACKS ontology (Premkumar et al. 2014)	Ashino's Materials Ontology, MatOnto	Laminated composites	Knowledge-guided design

^aDOLCE stands for Descriptive Ontology for Linguistic and Cognitive Engineering

^bEXPO ontology is used to describe scientific experiments

^cEurobau.com compiles construction materials data from ten European countries

^dGoodRelations ontology (Hepp 2008) is used for e-commerce with concepts such as business entities and prices

Big Semantic Data Processing in the Materials Design Domain, Table 2 Comparison of materials ontologies from a knowledge representation perspective

Materials ontology	Ontology metrics	Language	Modularity
Ashino's Materials Ontology (Ashino 2010)	606 concepts, 31 relationships, 488 instances	OWL	✓
Plinius ontology (van der Vet et al. 1994)	17 concepts, 4 relationships, 119 instances ^a	Ontolingua code	
MatOnto (Cheung et al. 2008)	78 concepts, 10 relationships, 24 instances	OWL	✓
PREMAP ontology (Bhat et al. 2013)	62 concepts	UML	✓
FreeClassOWL (Radinger et al. 2013)	5714 concepts, 225 relationships 1469 instances	OWL	
MatOWL (Zhang et al. 2009)	(not available)	OWL	
MMOY (Zhang et al. 2016)	544 metal concepts, 1781 related concepts, 9 relationships, 318 metal instances 1420 related instances	OWL	
ELSSI-EMD ontology (CEN 2010)	35 concepts, 37 relationships, 33 instances	OWL	✓
SLACKS ontology (Premkumar et al. 2014)	34 concepts and 10 relationships at least ^b	OWL	

^a103 instances out of 119 are elements in the periodic system

^bThe numbers are based on the high-level class diagram and an illustration of instances' integration in SLACKS shown in Premkumar et al. (2014)

Standards

There are currently not so many standards yet in this domain. Early efforts including ISO standards and MatML achieved limited adoption according to Austin (2016). The standard ISO 10303-45 includes an information model for materials properties. It provides schemas for materials properties, chemical compositions, and measure values (Swindells 2009). ISO 10303-235 includes an information model for product design and verification. MatML (Kaufman and Begley 2003, <https://www.matml.org/>) is an XML-based markup language for materials property data which includes schemas for such things as materials properties, composition, heat, and production.

Some other standards that have received more attention are, e.g., ThermoML and CML. ThermoML (Frenkel et al. 2006, 2011) is an XML-based markup language for exchange of thermo-physical and thermochemical property data. It covers over 120 properties regarding thermodynamic and transport property data for pure compounds, multicomponent mixtures, and chemical reactions. CML or Chemical Markup Language (Murray-Rust and Rzepa 2011; Murray-Rust et al. 2011) covers chemistry and especially molecules, reactions, solid state, computation, and spectroscopy. It is an extensible language that allows for the creation of sub-domains through the convention construct. Further, the dictionaries construct allows for connecting CML elements to dictionaries (or ontologies). This was inspired by the approach of the Crystallographic Information Framework or CIF (Bernstein et al. 2016, <http://www.iucr.org/resources/cif>).

The European Committee for Standardization (CEN) organized workshops on standards for materials engineering data (Austin 2016) of which the results are documented in CEN (2010). The work focuses specifically on ambient temperature tensile testing and developed schemas as well as an ontology (the ELSSI-EMD ontology from above).

Another recent approach is connected to the European Centre of Excellence NOMAD (Ghirghelli et al. 2016). The NOMAD repository's (<https://repository.nomad-coe.eu/>)

metadata structure is formatted to be independent of the electronic-structure theory or molecular-simulation code that was used to generate the data and can thus be used as an exchange format.

Conclusion

The use of the materials data in a materials design workflow requires addressing several big data problems including variety, variability, and veracity. Semantic technologies are a key factor in tackling some of these problems. Currently, efforts have started in creating materials databases, ontologies, and standards. However, much work remains to be done. To make full use of these resources, there is a need for integration of different kinds of resources and reasoning capabilities should be used, as in the bioinformatics field in the 1990s (Lambrix et al. 2009). Databases could use ontologies to define their schemas and enable ontology-based querying. Integration of databases is enabled by the use of ontologies. However, when databases have used different ontologies, alignments between different ontologies are needed as well (Euzenat and Shvaiko 2007). Further, more effort should be put on connecting ontologies and standards (as started in the CML, CEN, and NOMAD approaches), which may also lead to connections between different standards. Reasoning can be used in different ways. When developing resources reasoning can help in debugging and completing the resources leading to higher-quality resources (Ivanova and Lambrix 2013). Reasoning can also be used during querying of databases as well as in the process of connecting different resources.

References

- Agrawal A, Alok C (2016) Perspective: materials informatics and big data: realization of the fourth paradigm of science in materials science. *APL Mater* 4:053,208:1–10. <https://doi.org/10.1063/1.4946894>
- Ashino T (2010) Materials ontology: an infrastructure for exchanging materials information and knowledge. *Data Sci J* 9:54–61. <https://doi.org/10.2481/dsj.008-041>
- Austin T (2016) Towards a digital infrastructure for engineering materials data. *Mater Discov* 3:1–12. <https://doi.org/10.1016/j.md.2015.12.003>

- Belsky A, Hellenbrandt M, Karen VL, Luksch P (2002) New developments in the inorganic crystal structure database (ICSD): accessibility in support of materials research and design. *Acta Crystallogr Sect B Struct Sci* 58(3):364–369. <https://doi.org/10.1107/S0108768102006948>
- Bergerhoff G, Hundt R, Sievers R, Brown ID (1983) The inorganic crystal structure data base. *J Chem Inf Comput Sci* 23(2):66–69. <https://doi.org/10.1021/ci00038a003>
- Bernstein HJ, Bollinger JC, Brown ID, Grazulis S, Hester JR, McMahon B, Spadaccini N, Westbrook JD, Westrip SP (2016) Specification of the crystallographic information file format, version 2.0. *J Appl Cryst* 49:277–284. <https://doi.org/10.1107/S1600576715021871>
- Bhat M, Shah S, Das P, Reddy S (2013) Premλp: knowledge driven design of materials and engineering process. In: ICoRD'13. Springer, pp 1315–1329. https://doi.org/10.1007/978-81-322-1050-4_105
- Campbell CE, Kattner UR, Liu ZK (2014) File and data repositories for next generation CALPHAD. *Scr Mater* 70(Suppl C):7–11. <https://doi.org/10.1016/j.scriptamat.2013.06.013>
- Ceder G, Persson KA (2013) The Stuff of Dreams. *Sci Am* 309:36–40
- CEN (2010) A guide to the development and use of standards compliant data formats for engineering materials test data. European Committee for Standardization
- Cheng X, Hu C, Li Y (2014) A semantic-driven knowledge representation model for the materials engineering application. *Data Sci J* 13:26–44. <https://doi.org/10.2481/dsj.13-061/>
- Cheung K, Drennan J, Hunter J (2008) Towards an ontology for data-driven discovery of new materials. In: McGuinness D, Fox P, Brodaric B (eds) Semantic scientific knowledge integration AAAI/SSS workshop, pp 9–14
- Curtarolo S, Setyawan W, Wang S, Xue J, Yang K, Taylor R, Nelson L, Hart G, Sanvito S, Buongiorno-Nardelli M, Mingo N, Levy O (2012) AFLOWLIB.ORG: a distributed materials properties repository from high-throughput ab initio calculations. *Comput Mater Sci* 58(Supplement C):227–235. <https://doi.org/10.1016/j.commatsci.2012.02.002>
- Curtarolo S, Hart G, Buongiorno-Nardelli M, Mingo N, Sanvito S, Levy O (2013) The high-throughput highway to computational materials design. *Nat Mater* 12(3):191. <https://doi.org/10.1038/nmat3568>
- Euzenat J, Shvaiko P (2007) Ontology matching. Springer, Berlin/Heidelberg
- Faber F, Lindmaa A, von Lilienfeld A, Armiento R (2016) Machine learning energies of 2 million Elpasolite \$(AB{C})_{2}{D}_{6}\$ crystals. *Phys Rev Lett* 117(13):135,502. <https://doi.org/10.1103/PhysRevLett.117.135502>
- Frenkel M, Chirico RD, Diky V, Dong Q, Marsh KN, Dymond JH, Wakeham WA, Stein SE, Knigsberger E, Goodwin ARH (2006) XML-based IUPAC standard for experimental, predicted, and critically evaluated thermodynamic property data storage and capture (ThermoML) (IUPAC Recommendations 2006). *Pure Appl Chem* 78:541–612. <https://doi.org/10.1351/pac200678030541>
- Frenkel M, Chirico RD, Diky V, Brown PL, Dymond JH, Goldberg RN, Goodwin ARH, Heerklotz H, Knigsberger E, Ladbury JE, Marsh KN, Remeta DP, Stein SE, Wakeham WA, Williams PA (2011) Extension of ThermoML: the IUPAC standard for thermodynamic data communications (IUPAC recommendations 2011). *Pure Appl Chem* 83:1937–1969. <https://doi.org/10.1351/PAC-REC-11-05-01>
- Gangemi A, Guarino N, Masolo C, Oltramari A, Schneider L (2002) Sweetening ontologies with dolce. Knowledge engineering and knowledge management: ontologies and the semantic web, pp 223–233. https://doi.org/10.1007/3-540-45810-7_18
- Gaultois MW, Oliynyk AO, Mar A, Sparks TD, Mulholland GJ, Meredig B (2016) Perspective: web-based machine learning models for real-time screening of thermoelectric materials properties. *APL Mater* 4(5):053,213. <https://doi.org/10.1063/1.4952607>
- Ghiringhelli LM, Carbogno C, Levchenko S, Mohamed F, Huhs G, Lueders M, Oliveira M, Scheffler M (2016) Towards a common format for computational materials science data. *PSI-K Scientific Highlights July*
- Glasser L (2016) Crystallographic information resources. *J Chem Edu* 93(3):542–549. <https://doi.org/10.1021/acs.jchemed.5b00253>
- Grazulis S, Dazkevic A, Merkys A, Chateigner D, Lutterotti L, Quiros M, Serebryanaya NR, Moeck P, Downs RT, Le Bail A (2012) Crystallography open database (COD): an open-access collection of crystal structures and platform for world-wide collaboration. *Nucleic Acids Res* 40(Database issue):D420–D427. <https://doi.org/10.1093/nar/gkr900>
- Hepp M (2008) Goodrelations: an ontology for describing products and services offers on the web. *Knowl Eng Pract Patterns* 329–346. https://doi.org/10.1007/978-3-540-87696-0_29
- Ivanova V, Lambrix P (2013) A unified approach for debugging is-a structure and mappings in networked taxonomies. *J Biomed Semant* 4:10:1–10:19. <https://doi.org/10.1186/2041-1480-4-10>
- Jain A, Ong SP, Hautier G, Chen W, Richards WD, Dacek S, Cholia S, Gunter D, Skinner D, Ceder G, Persson KA (2013) Commentary: the materials project: a materials genome approach to accelerating materials innovation. *APL Mater* 1(1):011,002. <https://doi.org/10.1063/1.4812323>
- Kaufman JG, Begley EF (2003) MatML: a data interchange markup language. *Adv Mater Process* 161: 35–36
- Lambrix P, Strömbäck L, Tan H (2009) Information integration in bioinformatics with ontologies and standards. In: Bry F, Maluszynski J (eds) Semantic tech-

- niques for the Web, pp 343–376. https://doi.org/10.1007/978-3-642-04581-3_8
- Larsen AH, Mortensen JJ, Blomqvist J, Castelli IE, Christensen R, Duak M, Friis J, Groves MN, Hammer B, Hargus C, Hermes ED, Jennings PC, Jensen PB, Kermode J, Kitchin JR, Kolsbjergr EL, Kubal J, Kaasbjergr K, Lysgaard S, Maronsson JB, Maxson T, Olsen T, Pastewka L, Peterson A, Rostgaard C, Schitz J, Schtt O, Strange M, Thygesen KS, Vegge T, Vilhelmsen L, Walter M, Zeng Z, Jacobsen KW (2017) The atomic simulation environment – a Python library for working with atoms. *J Phys Condens Matter* 29(27):273,002. <https://doi.org/10.1088/1361-648X/aa680e>
- Lejaeghere K, Bihlmayer G, Bjrkman T, Blaha P, Blgels S, Blum V, Caliste D, Castelli IE, Clark SJ, Corso AD, Gironcoli Sd, Deutsch T, Dewhurst JK, Marco ID, Draxl C, Duak M, Eriksson O, Flores-Livas JA, Garritt KF, Genovese L, Giannozzi P, Giantomassi M, Goedecker S, Gonze X, Grns O, Gross EKU, Gulans A, Gygi F, Hamann DR, Hasnip PJ, Holzwarth NaW, Iuan D, Jochym DB, Jollet F, Jones D, Kresse G, Koepernik K, Kkbenli E, Kvashnin YO, Locht ILM, Lubeck S, Marsman M, Marzari N, Nitzsche U, Nordstrm L, Ozaki T, Paulatto L, Pickard CJ, Poelmans W, Probert MJ, Refson K, Richter M, Rignanese GM, Saha S, Scheffler M, Schlipf M, Schwarz K, Sharma S, Tavazza F, Thunstrm P, Tkatchenko A, Torrent M, Vanderbilt D, van Setten MJ, Speybroeck VV, Wills JM, Yates JR, Zhang GX, Cottenier S (2016) Reproducibility in density functional theory calculations of solids. *Science* 351(6280):aad3000. <https://doi.org/10.1126/science.aad3000>
- Moruzzi VL, Janak JF, Williams ARAR (2013) Calculated electronic properties of metals. Pergamon Press, New York
- Mulholland GJ, Paradiso SP (2016) Perspective: materials informatics across the product lifecycle: selection, manufacturing, and certification. *APL Mater* 4(5):053,207. <https://doi.org/10.1063/1.4945422>
- Murray-Rust P, Rzepa HS (2011) CML: evolution and design. *J Cheminf* 3:44. <https://doi.org/10.1186/1758-2946-3-44>
- Murray-Rust P, Townsend JA, Adams SE, Phadungsukanan W, Thomas J (2011) The semantics of chemical markup language (CML): dictionaries and conventions. *J Cheminfor* 3:43. <https://doi.org/10.1186/1758-2946-3-43>
- Pizzi G, Cepellotti A, Sabatini R, Marzari N, Kozinsky B (2016) AiiDA: automated interactive infrastructure and database for computational science. *Comput Mater Sci* 111(Supplement C):218–230. <https://doi.org/10.1016/j.commatsci.2015.09.013>
- Premkumar V, Krishnamurty S, Wileden JC, Grosse IR (2014) A semantic knowledge management system for laminated composites. *Adv Eng Inf* 28(1):91–101. <https://doi.org/10.1016/j.aei.2013.12.004>
- Radinger A, Rodriguez-Castro B, Stoltz A, Hepp M (2013) Baudataweb: the Austrian building and construction materials market as linked data. In: Proceedings of the 9th international conference on semantic systems. ACM, pp 25–32. <https://doi.org/10.1145/2506182.2506186>
- Rajan K (2015) Materials informatics: the materials Gene and big data. *Annu Rev Mater Res* 45:153–169. <https://doi.org/10.1146/annurev-matsci-070214-021132>
- Saal JE, Kirklin S, Aykol M, Meredig B, Wolverton C (2013) Materials design and discovery with high-throughput density functional theory: the open quantum materials database (OQMD). *JOM* 65(11):1501–1509. <https://doi.org/10.1007/s11837-013-0755-4>
- Soldatova LN, King RD (2006) An ontology of scientific experiments. *J R Soc Interface* 3(11):795–803. <https://doi.org/10.1098/rsif.2006.0134>
- Swindells N (2009) The representation and exchange of material and other engineering properties. *Data Sci J* 8:190–200. <https://doi.org/10.2481/dsj.008-007>
- van der Vet P, Speel PH, Mars N (1994) The Plinius ontology of ceramic materials. In: Mars N (ed) Workshop notes ECAI'94 workshop comparison of implemented ontologies, pp 187–205
- Vardeman C, Krisnadhi A, Cheatham M, Janowicz K, Ferguson H, Hitzler P, Buccellato A (2017) An ontology design pattern and its use case for modeling material transformation. *Semant Web* 8:719–731. <https://doi.org/10.3233/SW-160231>
- Zhang X, Hu C, Li H (2009) Semantic query on materials data based on mapping matml to an owl ontology. *Data Sci J* 8:1–17. <https://doi.org/10.2481/dsj.8.1>
- Zhang X, Zhao C, Wang X (2015a) A survey on knowledge representation in materials science and engineering: an ontological perspective. *Comput Ind* 73:8–22. <https://doi.org/10.1016/j.compind.2015.07.005>
- Zhang Y, Luo X, Zhao Y, chao Zhang H (2015b) An ontology-based knowledge framework for engineering material selection. *Adv Eng Inf* 29:985–1000. <https://doi.org/10.1016/j.aei.2015.09.002>
- Zhang X, Pan D, Zhao C, Li K (2016) MMOY: towards deriving a metallic materials ontology from Yago. *Adv Eng Inf* 30:687–702. <https://doi.org/10.1016/j.aei.2016.09.002>
- Zhang X, Chen H, Ruan Y, Pan D, Zhao C (2017) MATVIZ: a semantic query and visualization approach for metallic materials data. *Int J Web Inf Syst* 13:260–280. <https://doi.org/10.1108/IJWIS-11-2016-0065>

Big Spatial Data Access Methods

► Indexing

Bioinformatics

- ▶ [Big Data Analysis in Bioinformatics](#)
-

Bioinformatics Algorithms

- ▶ [Big Data Analysis in Bioinformatics](#)
-

Bioinformatics Big Data

- ▶ [Big Data Analysis in Bioinformatics](#)
-

Biomedical Linked Data

- ▶ [Big Semantic Data Processing in the Life Sciences Domain](#)
-

Blind Data Linkage

- ▶ [Privacy-Preserving Record Linkage](#)
-

Blockchain Consensus

- ▶ [Blockchain Transaction Processing](#)
-

Blockchain Data Management

- ▶ [Blockchain Transaction Processing](#)

Blockchain Transaction Processing

Suyash Gupta¹ and Mohammad Sadoghi²

¹Department of Computer Science, University of California, Davis, CA, USA

²University of California, Davis, CA, USA

Synonyms

Blockchain consensus; Blockchain data management; Cryptocurrency

Definitions

A blockchain is a linked list of immutable tamper-proof blocks, which is stored at each participating node. Each block records a set of transactions and the associated metadata. Blockchain transactions act on the identical ledger data stored at each node. Blockchain was first perceived by Satoshi Nakamoto (Satoshi 2008), as a peer-to-peer money exchange system. Nakamoto referred to the transactional tokens exchanged among clients in his system as Bitcoins.

Overview

In 2008, Satoshi Nakamoto (Satoshi 2008) came up with the design of an unanticipated technology that revolutionized the research in the distributed systems community. Nakamoto presented the design of a peer-to-peer money exchange process that was shared yet distributed. Nakamoto named his transactional token as *Bitcoin* and brought forth design of a new paradigm *blockchain*.

The key element of any blockchain system is the existence of an immutable tamper-proof *block*. In its simplest form, a block is an encrypted aggregation of a set of transactions. The existence of a block acts as a guarantee that the transactions have been executed and verified.

A newly created block is appended to an existing chain of blocks. This chain of blocks is predominantly a *linked list* which associates one block with the other. The initial block of any such list is a *genesis block* (Decker and Wattenhofer 2013). Genesis block is a special block that is numbered zero and is hard-coded in the blockchain application. Each other block links to some previously existing block. Hence, a blockchain grows by appending new blocks to the existing chain.

A transaction in a blockchain system is identical to any distributed or OLTP transaction (TPP Council 2010) that acts on some data. Traditional blockchain applications (such as Bitcoin) consist of transactions that represent an exchange of money between two entities (or users). Each valid transaction is recorded in a block, which can contain multiple transactions, for efficiency. Immutability is achieved by leveraging strong cryptographic properties such as hashing (Katz and Lindell 2007). Figure 1 presents the structure of a simple blockchain.

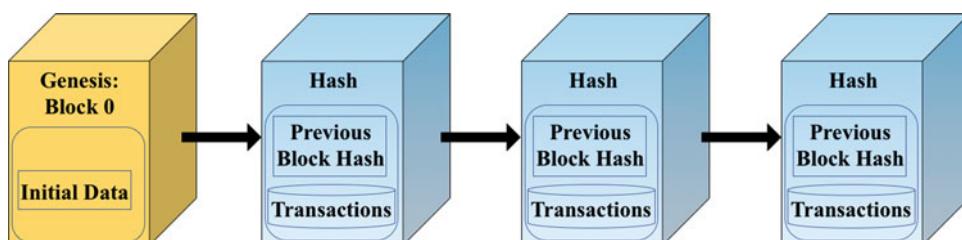
A blockchain is a *linked list* in a true sense, as each block stores the hash of the previous block in its chain. Each block also digitally signs its contents by storing the hash of its contents inside the block. These hashes provide cryptographic integrity, as any adversary intending to modify a block needs to also modify all the previous blocks in a chain, which makes the attack cryptographically infeasible. A key design strategy is to construct a Merkle tree (Katz and Lindell 2007) to efficiently store and verify the hashes. Thus, each block only stores the *root* of the Merkle tree, as given the root, it is easy to verify the immutability.

The preceding discussion allows us to summarize that a blockchain aims at securely storing a set of transactions. In the succeeding sections, we discuss in detail the transaction processing in a blockchain system. We also study mechanisms to validate these transactions and analyze some blockchain applications that employ the same.

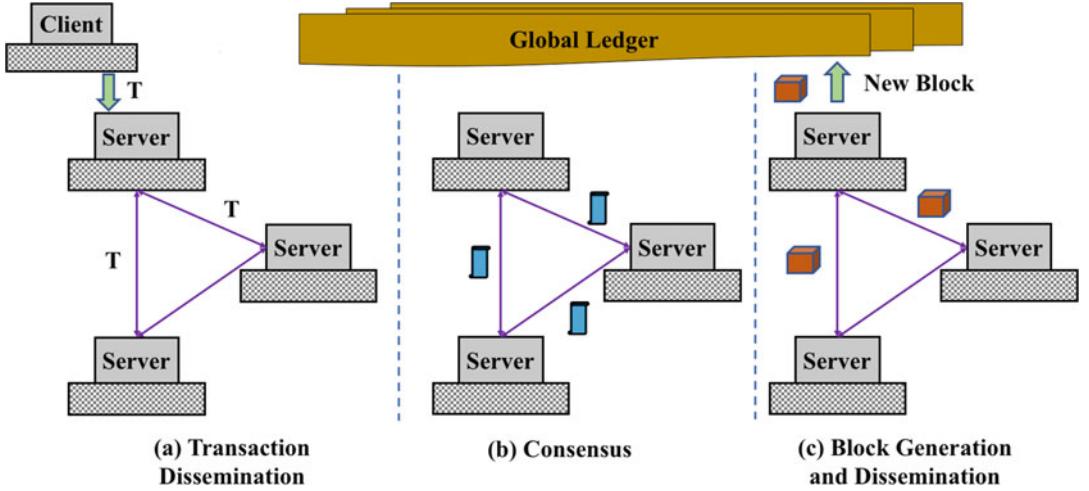
Key Research Findings

Transactions in a blockchain system are identical to their traditional database counterparts. These transactions are issued by the clients to the servers of the blockchain system (Nawab 2018). These transactions act on the data stored on all the participating servers. In its vanilla form, a blockchain transaction could be visualized as a set of *read/write* operations performed on each node of a replicated distributed database. To determine an ordering for all the incoming transactions, each blockchain application employs a consensus (Steen and Tanenbaum 2017) protocol.

A distributed consensus algorithm (Lamport 1998; Gray and Lamport 2006) allows a system to reach a common decision, respected by the majority of nodes. Recent blockchain technologies present several strategies for establishing consensus: Proof-of-Work (Jakobsson and Juels 1999; Satoshi 2008), Proof-of-Stake (King and Nadal 2012), Proof-of-Authority (Parity Technologies 2018), Practical Byzantine Fault Tolerance (Castro and Liskov 1999; Cachin 2016), and so on. To quantify the allowed set of nodes that can create a block (or participate in the consensus process), it is also necessary to characterize the topologies for blockchain systems.



Blockchain Transaction Processing, Fig. 1 Basic blockchain representations



Blockchain Transaction Processing, Fig. 2

Blockchain flow: Three main phases in any blockchain application are represented. (a) Client sends a transaction to one of the servers, which it disseminates to all the

other servers. (b) Servers run the underlying consensus protocol, to determine the block creator. (c) New block is created and transmitted to each node, which also implies adding to global ledger

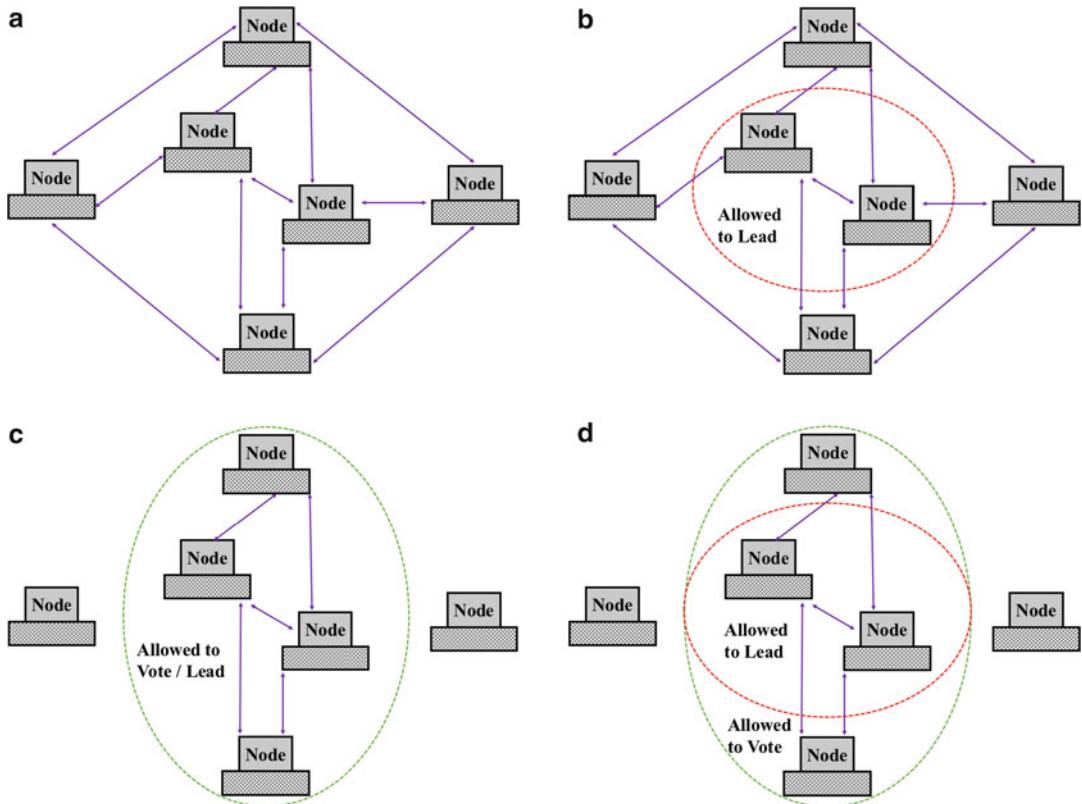
Blockchain Execution

Figure 2 illustrates the three main phases required by any blockchain application to create a new block. The client transmits a transactional request to one of the servers. This server multicasts the request to all other servers. We term this phase as *transaction dissemination*. Once all the servers have a copy of client request, they initiate a *consensus* protocol. The choice of underlying consensus protocol affects the time complexity and resource consumption. The winner of the consensus phase generates the next block and transmits it to all other servers. This transmission process is equivalent to adding an entry (block) to the global distributed ledger.

Blockchain Topologies

A key parameter that renders the design of a blockchain application is the categorization of nodes to be part of the application. Recent works (Pilkington 2015; Cachin and Vukolic 2017) categorize blockchain systems as either public, private, permissioned, or hybrid. Although the characteristics of a public blockchain is clearly stated in the community, there is no common consensus on the terminology pertaining to the other terms.

We categorize blockchain systems under four heads: public, private, permissioned, and hybrid. Figure 3 presents a pictorial representation of the different categories. In these figures nodes that are not allowed to participate in the activities pertaining to the network lack connections to any node in the network. We use different circles to demarcate different zones of operation; certain nodes are allowed to lead (or create the block), and some are allowed to participate in the consensus protocol. *Public blockchain* systems such as Bitcoin (Satoshi 2008) and Ethereum (Wood 2015) allow any node to participate in the consensus process, and any node can generate the next valid block. Hence, if all the nodes have same resources, then each node has equal probability of creating a block (We are assuming fair and ideal conditions, where each node works independently.). *Private blockchain* systems are at the other end of the spectrum, as they allow only some nodes to be part of the consensus process, and only a subset of these nodes can generate the next block. These systems could be utilized at a banking organization which only allows its customers to participate in the consensus, while its employees are only permitted to commit the results by creating a block.



Blockchain Transaction Processing, Fig. 3 Topologies for blockchain systems. (a) Public blockchain. (b) Hybrid blockchain. (c) Permissioned blockchain. (d) Private blockchain

Hybrid blockchain systems attain a middle ground between the two extremes. These systems allow any node to be part of the consensus process, but only some designated nodes are allowed to form the next block. Cryptocurrency Ripple (Schwartz et al. 2014) supports a variant of the hybrid model, where some public institutions can act as transaction validators. *Permissioned blockchain* systems are the restricted variant of public blockchain systems. These systems enlist few nodes as the members with equal rights, that is, each node as part of the system can form a block. These systems allow us to represent internal working of any organization, social media groups, and an early stage startup.

Blockchain Consensus

The common denominator for all the blockchain applications is the underlying distributed consensus algorithm. Nakamoto suggested **Proof-**

of-Work (Satoshi 2008) (henceforth referred as PoW) for achieving consensus among the participating nodes. PoW requires each node to demonstrate its ability to solve a nontrivial task. The participating nodes compete among themselves to solve a complex puzzle (such as computation of a 256-bit SHA value). The node which computes the solution gets the opportunity to generate the next *block*. It also disseminates the block to all the nodes, which marks as the *proof* that it performed some nontrivial computation. Hence, all other nodes respect the winner's ability and reach the consensus by continuing to chain ahead of this block.

In PoW the accessibility to larger computational resources determines the winner node(s). However, it is possible that multiple nodes could lay claim for the next block to be added to the chain. Based on the dissemination of the new block, this could lead to *branching* out of the

single chain. Interestingly, these branches are often short-lived, as all the nodes tend to align themselves to the longest chain, which in turn leads to pruning of the branches. It is important to understand that a node receives incentive when it is able to append a block to the longest chain. Hence, it is to their advantage to always align with the longest chain; otherwise they would end up spending their computational resources for zero gains. Note: PoW algorithm can theoretically be compromised by an adversary controlling at least 51% of computational resources in the network. This theoretical possibility has practical implications as a group of miners can share resources to generate blocks faster, skewing the decentralized nature of the network.

Proof-of-Stake (King and Nadal 2012) (henceforth referred at PoS) aims at preserving the decentralized nature of the blockchain network. In PoS algorithm a node with $n\%$ resources gets $n\%$ time opportunity to create a block. Hence, the underlying principle in PoS is that the node with higher stake lays claim to the generation of the next block. To determine a node's stake, a combination of one or more factors, such as wealth, resources, and so on, could be utilized. PoS algorithm requires a set of nodes to act as validators. Any node that wants to act as a validator needs to *lock out* its resources, as a proof of its stake.

PoS only permits a validator to create new blocks. To create a new block, a set of validators participate in the consensus algorithm. PoS consensus algorithm has two variants: (i) chain-based and (ii) BFT-style. *Chain-based* PoS algorithm uses a pseudorandom algorithm to select a validator, which then creates a new block and adds it to the existing chain of blocks. The frequency of selecting the validator is set to some predefined time interval. *BFT-style* algorithm runs a byzantine fault tolerance algorithm to select the next valid block. Here, validators are given the right to propose the next block, at random. Another key difference between these algorithms is the synchrony requirement; *chain-based* PoS algorithms are inherently synchronous, while *BFT-style* PoS is partially synchronous.

Early implementations of PoS such as Peercoin (King and Nadal 2012) suffer from non-convergence, that is, lack of single chain (also referred as “nothing-at-stake” attack). Such a situation happens when the longest chain branches into multiple forks. Intuitively, the nodes should aim at converging the branches into the single longest chain. However, if the participating nodes are incentive driven, then they could create blocks in both the chains, to maximize their gains. Such an attack is not possible on PoW as it would require a node to have double the number of computational resources, but as PoS does not require solving a complex problem, nodes can easily create multiple blocks.

Proof-of-Authority (Parity Technologies 2018) (henceforth referred as PoA) is designed to be used alongside nonpublic blockchain systems. The key idea is to designate a set of nodes as the authority. These nodes are entrusted with the task of creating new blocks and validating the transactions. PoA marks a block as part of the blockchain if it is signed by majority of the authorized nodes. The incentive model in PoA highlights that it is in the interest of an authority node to maintain its reputation, to receive periodic incentives. Hence, PoA does not select nodes based on their claimed stakes.

Proof-of-Space (Ateniese et al. 2014; Dziembowski et al. 2015) also known as *Proof-of-Capacity* (henceforth referred as PoC) is a consensus algorithm orthogonal to PoW. It expects nodes to provide a proof that they have sufficient “storage” to solve a computational problem. PoC algorithm targets computational problems such as *hard-to-pebble graphs* (Dziembowski et al. 2015) that need large amount of memory storage to solve the problem. In the PoC algorithm, the verifier first expects a prover to commit to a labeling of the graph, and then he queries the prover for random locations in the committed graph. The key intuition behind this approach is that unless the prover has sufficient storage, he would not pass the verification. PoC-based cryptocurrency such as SpaceMint (Park et al. 2015) claims PoC-based approaches more resource efficient to PoW as storage consumes less energy.

A decade prior to the first blockchain application, distributed consensus problem had excited the researchers. Paxos (Lamport 1998) and Viewstamped Replication (Oki and Liskov 1988) presented key solutions to distributed consensus when the node failures were restricted to *fail-stop*. Castro and Liskov (1999) presented their novel *Practical Byzantine Fault Tolerance* (henceforth referred as PBFT) consensus algorithm to handle byzantine failures. Interestingly, all of previously discussed consensus algorithms provide similar guarantees as PBFT. Garay et al. (2015) helped in establishing an equivalence between the PoW and general consensus algorithms. This equivalence motivated the community to design efficient alternatives to PoW.

PBFT runs a three-phase protocol to reach a consensus among all the non-byzantine nodes. PBFT requires a bound on the number of byzantine nodes. If “ n ” represents the total number of nodes in the system, then PBFT bounds the number of byzantine nodes “ f ” as $f \leq \lfloor \frac{n+1}{3} \rfloor$; PBFT-based algorithms have an advantage of reduced resource consumption but suffer from large message complexity (order $O(n^2)$). Hence, these algorithms are preferred for restricted or small blockchain systems, in order to have less message overhead.

The algorithm starts with a client sending a *request* to the primary node. The primary node verifies the request, assigns a sequence number, and sends a *pre-prepare* message to all the replicas. Each *pre-prepare* message also contains the client’s *request* and the current *view* number. When a replica receives a *pre-prepare* message, it first verifies the request and then transmits a *prepare* message to all the nodes. The *prepare* message contains the digest of client *request*, sequence number, and *view* number. Once a node receives $2f$ *prepare* messages, matching to the *pre-prepare* message, it multicasts a *commit* message. The *commit* message also contains the digest of client *request*, sequence number, and *view* number. Each replica waits for receiving identical $2f+1$ *commit* messages, before executing the request and then transmits the solution to the client.

The client needs $f + 1$ matching responses from different replicas, to mark the request as complete. If the client *timeouts* while waiting for $f + 1$ responses, then it multicasts the request to all the replicas. Each replica on receiving a request from the client, which it has not executed, starts a timer and queries the primary. If the primary does not reply until *timeout*, then the replicas proceed to change the primary. Note: PBFT heavily relies on strong cryptographic guarantees, and each message is digitally signed by the sender using his private key.

Zyzzyva (Kotla et al. 2007) is another interesting byzantine fault tolerance protocol aimed at reducing the costs associated with BFT-style protocols. Zyzzyva (Kotla et al. 2007) allows the replicas to reach *early consensus* by permitting speculative execution. In Zyzzyva, the replicas are relieved from the task of ensuring a global order for the client requests. This task is delegated to the client, which in turn informs the replicas if they are not in sync.

The protocol initiates with a client sending a request to the primary, which in turn sends a *pre-prepare* message to all the replicas. The structure of the *prepare* message is similar to its PBFT counterpart except that it includes a digest summarizing the *history*. Each replica, on receiving a request from the primary, executes the request and transmits the response to the client. Each response, in addition to the fields in the *pre-prepare* message, contains the digest of history and a signed copy of message from the primary. If the client receives $3f + 1$ matching responses, then it assumes the response is stable and considers the request completed. In case the number of matching responses received by the client are in the range $2f + 1$ and $3f$, then it creates a *commit certificate* and transmits this certificate to all the nodes. The *commit certificate* includes the history which proves to a receiving replica that it can safely commit the history and start processing the next request. All the replicas acknowledge the client for the *commit certificate*. If the client receives less than $2f + 1$ responses, then it starts a timer and informs all the replicas again about the impending request. Now, either

the request would be safely completed, or a new primary would be elected.

Although Zyzzyva achieves higher throughput than PBFT, its *client-based* speculative execution is far from ideal. Zyzzyva places a lot of faith on the correctness of the client, which is quite discomforting, as a malicious client can prevent the replicas from maintaining linearizability. **Aardvark** (Clement et al. 2009) studies the ill-effects of various fast, byzantine fault-tolerant protocols and presents the design of a *robust* BFT protocol. In Aardvark, the messages are signed by the client using both digital signatures and message authentication codes (Katz and Lindell 2007). This prevents malicious clients from performing a *denial-of-service* attack, as it is costly for the client to sign each message two times. Aardvark uses *point-to-point* communication, instead of multicast communication. The key intuition behind such a choice is to disallow a faulty client of replica from blocking the complete network. Aardvark also periodically changes the primary replica. Each replica tracks the throughput of the current primary and suggests replacing the primary, when there is a decrease in its throughput. The authors use a simple timer to track the rate of primary response.

RBFT (Aublin et al. 2013) is a simple extension to Aardvark. RBFT aims at detecting *smartly* malicious primary replica, which could avoid being detected malicious by other replicas. The key intuition behind RBFT is to prevent the primary from insinuating delays that are within the stated threshold. Hence, the primary could reduce the throughput without being ever replaced. To tackle this problem, RBFT insists running $f + 1$ independent instances of the Aardvark protocol on each node. One of the instances is designated as the “master instance,” and it executes the requests. The rest of the instances are labeled as the “backup instances,” and they order the requests and monitor the performance of the master instance. If any backup instance observes a degradation of the performance of the master instance, then it transmits a message to elect a new primary. Note: RBFT does not allow more than one primary on any node. Hence, each node can have at most one primary instance. RBFT

protocol has an additional step in comparison to the three phases in PBFT and Aardvark. The client node initiates the protocol by transmitting a request to all the nodes. Next, each node propagates this request to all other nodes, and then the three-phase protocol begins. This extra round of redundancy ensures that the client request reaches all the instances.

Blockchain Systems

Bitcoin (Satoshi 2008) is regarded as the first blockchain application. It is a cryptographically secure digital currency with the aim of disrupting the traditional, institutionalized monetary exchange. Bitcoin acts as the token of transfer between two parties undergoing a monetary transaction. The underlying blockchain system is a network of nodes (also known as *miners*) that take a set of client transactions and validate the same by demonstrating a *Proof-of-Work*, that is, generating a block. The process of generating the next block is nontrivial and requires large computational resources. Hence, the miners are given incentives (such as Bitcoins) for dedicating their resources and generating the block. Each miner maintains locally an updated copy of the complete blockchain and the associated ledgers for every Bitcoin user.

To ensure Bitcoin system remains fair toward all the machines, the difficulty of *Proof-of-Work* challenge is periodically increased. We also know that Bitcoin is vulnerable to 51% attack, which can lead to *double spending* (Rosenfeld 2014). The intensity of such attacks increases when multiple forks of the longest chain are created. To avoid these attacks, Bitcoin developers suggest the clients to wait for their block to be confirmed before they mark the Bitcoins as transferred. This wait ensures that the specific block is a little deep (nearly six blocks) in the longest chain (Rosenfeld 2014). Bitcoin critics also argue that its *Proof-of-Work* consumes huge energy (As per some claims, one Bitcoin transaction consumes power equivalent to that required by 1.5 American homes per day.) and may not be a viable solution for the future.

Bitcoin-NG (Eyal et al. 2016) is a scalable variant to Bitcoin’s protocol. Our preceding discussion highlights that Bitcoin suffers from throughput degradation. This can be reduced by either increasing the block size or decreasing the block interval. However, increasing the former increases the propagation delays, and the latter can lead to incorrect consensus. Bitcoin-NG solves this problem by dividing the time into a set of intervals and selecting one leader for each time interval. The selected leader autonomously orders the transactions. Bitcoin-NG also uses two different block structures: *key blocks* for facilitating leader election and *microblocks* for maintaining the ledger.

Ethereum (Wood 2015) is a blockchain framework that permits users to create their own applications on top of the Ethereum Virtual Machine (EVM). Ethereum utilizes the notion of smart contracts to facilitate development of new operations. It also supports a digital cryptocurrency, *ether*, which is used to incentivize the developers to create correct applications. One of the key advantages of Ethereum is that it supports a Turing complete language to generate new applications on top of EVM. Initially, Ethereum’s consensus algorithm used the key elements of both PoW and PoC algorithms. Ethereum made nodes solve challenges that were not only compute intensive but also memory intensive. This design prevented existence of miners who utilized specially designed hardware for compute-intensive applications.

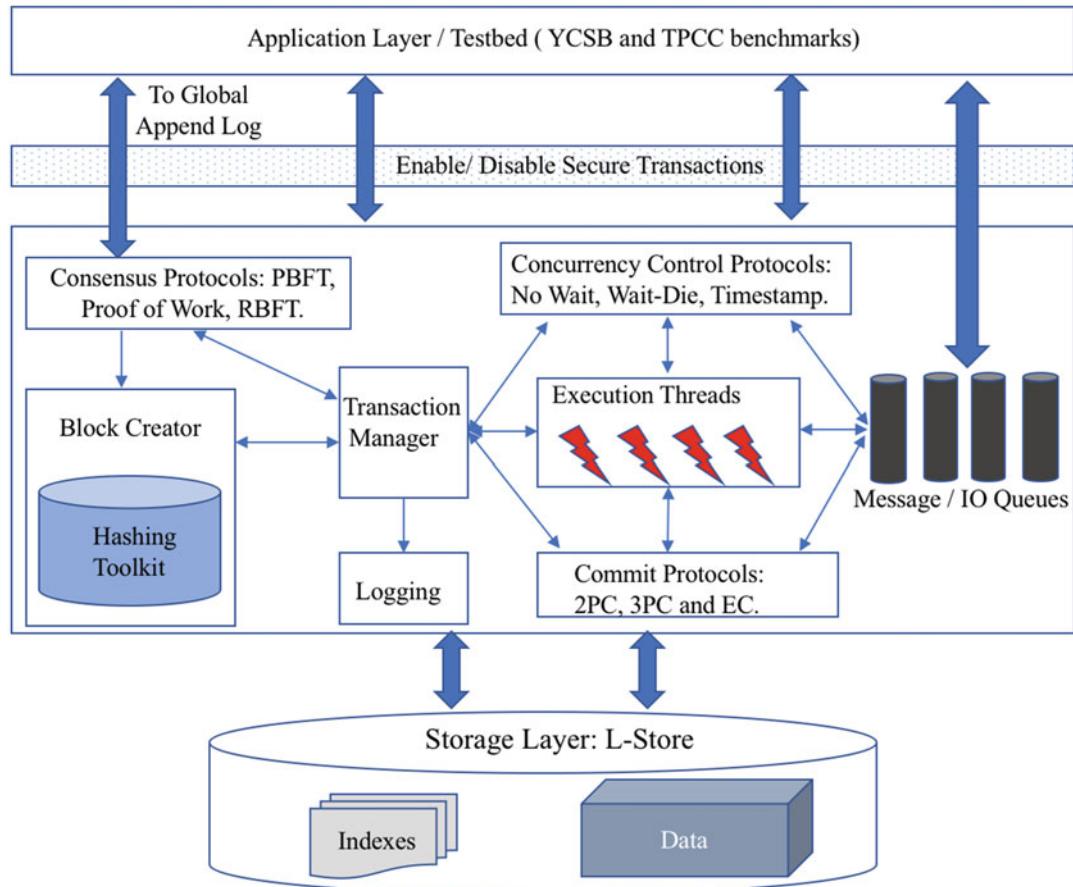
Recently, Ethereum modified its consensus protocol to include some notions of PoS algorithm. The modified protocol is referred as *Casper* (Buterin and Griffith 2017). Casper introduces the notion of *finality*, that is, it ensures that one chain becomes permanent in time. It also introduces the notion of *accountability*, that is, to penalize a validator, who performs the “nothing-at-stake” attack in PoS-based systems. The penalty leveraged on such a validator is equivalent to negating all his stakes.

Parity (Parity Technologies 2018) is an application designed on top of Ethereum. It provides an interface for its users to interact with

the Ethereum blockchain. Parity can be regarded as an interesting application for the blockchain community, as it provides support for both *Proof-of-Work* and *Proof-of-Authority* consensus algorithms. Hence, they allow mechanism for users of their application to set up “authority” nodes and resort to non-compute-intensive, POA algorithm.

Ripple (Schwartz et al. 2014) is considered as the third largest cryptocurrency after Bitcoin and Ethereum, in terms of market cap. It uses a consensus algorithm which is a simple variant of BFT algorithms. Ripple requires a number of failures f to be bounded as follows: $\leq (n - 1)/5 + 1$, where n represents the total number of nodes. Ripple’s consensus algorithm introduces the notion of a *Unified Node List* (UNL), which is a subset of the network. Each server communicates with the nodes in its UNL for reaching a consensus. The servers exchange the set of transactions they received from the clients and propose those transactions to their respective UNL for vote. If a transaction receives 80% of the votes, it is marked permanent. It is important to understand that if the generated UNL groups are a clique, then forks of the longest chain could coexist. Hence, UNLs are created in a manner that they share some set of nodes. Another noteworthy observation about Ripple protocol is that each client needs to select a set of validators or unique nodes that they trust. These validators next utilize Ripple consensus algorithm to verify the transactions.

Hyperledger (Cachin 2016) is a suite of resources aimed at modeling industry standard blockchain applications. It provides a series of application programming interfaces (APIs) for developers to create their own nonpublic blockchain applications. Hyperledger provides implementations of blockchain systems that uses RBFT and other variants of the PBFT consensus algorithm. It also facilitates the use and development of smart contracts. It is important to understand that the design philosophy of Hyperledger leans toward blockchain applications that require existence of nonpublic networks, and so, they do not need a compute-intensive consensus.



Blockchain Transaction Processing, Fig. 4 Architecture of ExpoDB

ExpoDB (Sadoghi 2017; Gupta and Sadoghi 2018) is an experimental research platform that facilitates design and testing of emerging database technologies. ExpoDB consists of a set of five layers providing distinct functionalities (refer to Fig. 4). *Application layer* acts as the test bed for evaluating the underlying database protocols (Gupta and Sadoghi 2018). It allows the system to access OLTP benchmarks such as YCSB Cooper et al. (2010), TPC-C TPP Council (2010), and PPS Harding et al. (2017). *Application layer* acts as an interface for the client-server interaction. *Transport layer* allows communication of messages between the client and server nodes.

Execution layer facilitates seamless execution of a transaction with the help of a set of threads.

These threads lie at the core of the execution layer as they run the transaction, abide by the rules of stated concurrency protocol, and achieve agreement among different transactional partitions. ExpoDB provides implementation for eight concurrency controls and three commit protocols. ExpoDB also characterizes a *storage layer* (Sadoghi et al. 2018) for storing the transactional data, messages, and relevant metadata.

ExpoDB extends blockchain functionality to the traditional distributed systems through a *secure layer*. To facilitate secure transactions, ExpoDB provides a cryptographic variant to YCSB – *Secure YCSB* benchmark. ExpoDB also contains implementations for a variety of consensus protocols such as PoW, PBFT, RBFT, and Bitcoin-NG.

Future Directions for Research

Although blockchain technology is just a decade old, it gained majority of its momentum in the last 5 years. This allows us to render different elements of the blockchain systems and achieve higher performance and throughput. Some of the plausible directions to develop efficient blockchain systems are as follows: (i) reducing the communication messages, (ii) defining efficient block structure, (iii) improving the consensus algorithm, and (iv) designing secure lightweight cryptographic functions.

Statistical and machine learning approaches have presented interesting solutions to automate key processes such as face recognition (Zhao et al. 2003), image classification (Krizhevsky et al. 2012), speech recognition (Graves et al. 2013), and so on. The tools can be leveraged to facilitate easy and efficient consensus. The intuition behind this approach is to allow learning algorithms to select nodes, which are fit to act as a block creator and prune the rest from the list of possible creators. The key observation behind such a design is that the nodes selected by the algorithm are predicted to be non-malicious. Machine learning techniques can play an important role in eliminating the human bias and inexperience. To learn which nodes can act as block creators, a feature set, representative of the nodes, needs to be defined. Some interesting features can be geographical distance, cost of communication, available computational resources, available memory storage, and so on. These features would help in generating the dataset that would help to train and test the underlying machine learning model. This model would be ran against new nodes that wish to join the associated blockchain application.

The programming languages and software engineering communities have developed several works that provide semantic guarantees to a language or an application (Wilcox et al. 2015; Leroy 2009; Kumar et al. 2014). These works have tried to formally verify (Keller 1976; Leroy 2009) the system using the principles of programming languages and techniques such as fi-

nite state automata, temporal logic, and model checking (Grumberg and Long 1994; Baier and Katoen 2008). We believe similar analysis can be performed in the context of blockchain applications. Theorem provers (such as Z3 De Moura and Bjørner 2008) and proof assistants (such as COQ Bertot 2006) could prove useful to define a certified blockchain application. A certified blockchain application can help in stating theoretical bounds on the resources required to generate a block. Similarly, some of the blockchain consensus has been shown to suffer from denial-of-service attacks (Bonneau et al. 2015), and a formally verified blockchain application can help realize such guarantees, if the underlying application provides such a claim.

References

- Ateniese G, Bonacina I, Faonio A, Galesi N (2014) Proofs of space: when space is of the essence. In: Abdalla M, De Prisco R (eds) Security and cryptography for networks. Springer International Publishing, pp 538–557
- Aublin PL, Mokhtar SB, Quéma V (2013) RBFT: redundant byzantine fault tolerance. In: Proceedings of the 2013 IEEE 33rd international conference on distributed computing systems, ICDCS ’13. IEEE Computer Society, pp 297–306
- Baier C, Katoen JP (2008) Principles of model checking (representation and mind series). The MIT Press, Cambridge/London
- Bertot Y (2006) Coq in a hurry. CoRR abs/cs/0603118, <http://arxiv.org/abs/cs/0603118>, cs/0603118
- Bonneau J, Miller A, Clark J, Narayanan A, Kroll JA, Felten EW (2015) SoK: research perspectives and challenges for bitcoin and cryptocurrencies. In: Proceedings of the 2015 IEEE symposium on security and privacy, SP ’15. IEEE Computer Society, Washington, DC, pp 104–121
- Buterin V, Griffith V (2017) Casper the friendly finality gadget. CoRR abs/1710.09437, <http://arxiv.org/abs/1710.09437>, 1710.09437
- Cachin C (2016) Architecture of the hyperledger blockchain fabric. In: Workshop on distributed cryptocurrencies and consensus ledgers, DCCL 2016
- Cachin C, Vukolic M (2017) Blockchain consensus protocols in the wild. CoRR abs/1707.01873
- Castro M, Liskov B (1999) Practical byzantine fault tolerance. In: Proceedings of the third symposium on operating systems design and implementation, OSDI ’99. USENIX Association, Berkeley, pp 173–186
- Clement A, Wong E, Alvisi L, Dahlin M, Marchetti M (2009) Making byzantine fault tolerant systems tolerate

- byzantine faults. In: Proceedings of the 6th USENIX symposium on networked systems design and implementation, NSDI'09. USENIX Association, Berkeley, pp 153–168
- Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing. ACM, pp 143–154
- Decker C, Wattenhofer R (2013) Information propagation in the bitcoin network. In: 13th IEEE international conference on peer-to-peer computing (P2P), Trento
- De Moura L, Bjørner N (2008) Z3: an efficient SMT solver. Springer, Berlin/Heidelberg/Berlin, pp 337–340
- Dziembowski S, Faust S, Kolmogorov V, Pietrzak K (2015) Proofs of space. In: Gennaro R, Robshaw M (eds) Advances in cryptology – CRYPTO 2015. Springer, Berlin/Heidelberg, pp 585–605
- Eyal I, Gencer AE, Sirer EG, Van Renesse R (2016) Bitcoin-NG: a scalable blockchain protocol. In: Proceedings of the 13th USENIX conference on networked systems design and implementation, NSDI'16. USENIX Association, Berkeley, pp 45–59
- Garay J, Kiayias A, Leonardos N (2015) The bitcoin backbone protocol: analysis and applications. Springer, Berlin/Heidelberg/Berlin, pp 281–310
- Graves A, Mohamed A, Hinton GE (2013) Speech recognition with deep recurrent neural networks. CoRR abs/1303.5778, <http://arxiv.org/abs/1303.5778>, 1303.5778
- Gray J, Lamport L (2006) Consensus on transaction commit. ACM TODS 31(1):133–160
- Grumberg O, Long DE (1994) Model checking and modular verification. ACM Trans Program Lang Syst 16(3):843–871
- Gupta S, Sadoghi M (2018) EasyCommit: a non-blocking two-phase commit protocol. In: Proceedings of the 21st international conference on extending database technology, Open Proceedings, EDBT
- Harding R, Van Aken D, Pavlo A, Stonebraker M (2017) An evaluation of distributed concurrency control. Proc VLDB Endow 10(5):553–564
- Jacobsson M, Juels A (1999) Proofs of work and bread pudding protocols. In: Proceedings of the IFIP TC6/TC11 joint working conference on secure information networks: communications and multimedia security, CMS '99. Kluwer, B.V., pp 258–272
- Katz J, Lindell Y (2007) Introduction to modern cryptography. Chapman & Hall/CRC, Boca Raton
- Keller RM (1976) Formal verification of parallel programs. Commun ACM 19(7):371–384
- King S, Nadal S (2012) PPCoin: peer-to-peer cryptocurrency with proof-of-stake. peercoin.net
- Kotla R, Alvisi L, Dahlin M, Clement A, Wong E (2007) Zyzzyva: speculative byzantine fault tolerance. In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP '07. ACM, New York, pp 45–58. <https://doi.org/10.1145/1294261.1294267>
- Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems, NIPS'12, vol 1. Curran Associates Inc., pp 1097–1105
- Kumar R, Myreen MO, Norrish M, Owens S (2014) CakeML: a verified implementation of ML. In: Proceedings of the 41st ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL '14. ACM, New York, pp 179–191
- Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169
- Leroy X (2009) A formally verified compiler back-end. J Autom Reason 43(4):363–446
- Nawab F (2018) Geo-scale transaction processing. Springer International Publishing, pp 1–7. https://doi.org/10.1007/978-3-319-63962-8_180-1
- Oki BM, Liskov BH (1988) Viewstamped replication: a new primary copy method to support highly-available distributed systems. In: Proceedings of the seventh annual ACM symposium on principles of distributed computing, PODC '88. ACM, New York, pp 8–17
- Parity Technologies (2018) Parity ethereum blockchain. <https://wwwparity.io/>
- Park S, Kwon A, Fuchsbaier G, Gaži P, Alwen J, Pietrzak K (2015) SpaceMint: a cryptocurrency based on proofs of space. <https://eprint.iacr.org/2015/528>
- Pilkington M (2015) Blockchain technology: principles and applications. In: Research handbook on digital transformations. SSRN
- Rosenfeld M (2014) Analysis of hashrate-based double spending. CoRR abs/1402.2009, <http://arxiv.org/abs/1402.2009>, 1402.2009
- Sadoghi M (2017) Expodb: an exploratory data science platform. In: Proceedings of the eighth biennial conference on innovative data systems research, CIDR
- Sadoghi M, Bhattacherjee S, Bhattacharjee B, Canim M (2018) L-store: a real-time OLTP and OLAP system. OpenProceeding.org, EDBT
- Satoshi N (2008) Bitcoin: a peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>
- Schwartz D, Youngs N, Britto A (2014) The ripple protocol consensus algorithm. <https://www.ripple.com/>
- Steen Mv, Tanenbaum AS (2017) Distributed systems, 3rd edn, version 3.01. distributed-systems.net
- TPC Council (2010) TPC benchmark C (Revision 5.11)
- Wilcox JR, Woos D, Panckehka P, Tatlock Z, Wang X, Ernst MD, Anderson T (2015) Verdi: a framework for implementing and formally verifying distributed systems. In: Proceedings of the 36th ACM SIGPLAN conference on programming language design and implementation, PLDI '15. ACM, New York, pp 357–368
- Wood G (2015) Ethereum: a secure decentralised generalised transaction ledger. <http://gavwood.com/paper.pdf>
- Zhao W, Chellappa R, Phillips PJ, Rosenfeld A (2003) Face recognition: a literature survey. ACM Comput Surv 35(4):399–458. <https://doi.org/10.1145/954339.954342>

Blotter Design and Protocols

- ▶ Achieving Low Latency Transactions for Geo-replicated Storage with Blotter

BSP Programming Model

Zuhair Khayyat

Lucidya LLC, Jeddah, Saudi Arabia

Synonyms

Bulk synchronous parallel; BSPlib

Definitions

It is an iterative share-nothing parallel computing model that abstracts the parallelization of applications into three stages: computation, communication, and synchronization. As illustrated in Fig. 1, each iteration, or superstep, runs the three steps in order and repeats them until the application completes its calculations. The input of the first iteration is loaded from the user's input dataset. After that, the input for each subsequent iteration is collected from the previous one.

Overview

This model is first proposed by Valiant (1990) as a generic parallel programming concept that is not bound to a specific to a programming model or hardware. There have been several modifications to the original design of BSP, such as extended BSP (Juurlink and Wijshoff 1996), decomposable BSP (Beran 1999), oblivious BSP (Gonzalez et al. 2000), CREW BSP (Goodrich 1999), heterogeneous BSP (Williams and Parsons 2000), and (d,x)-BSP (Blelloch et al. 1995). These variations try to improve the communication cost, increase data locality, and reduce synchronizations costs

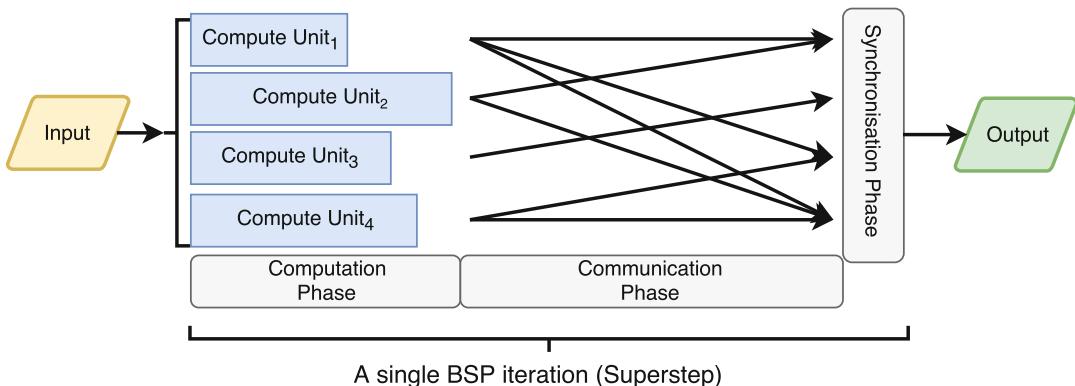
by subdividing the BSP system into smaller independent systems that run concurrently. BSP can be used for various applications including point-to-point message passing between different entities, fast Fourier transforms, data sorting, PageRank, and graph coloring. Although MPI may be used to implement BSP, several communication libraries, such as BSPlib (Hill et al. 1998), have been introduced to support the BSP model with customized message passing protocols naively.

Computation

This phase executes the user's logic. It generates multiple independent computing units from the user's code and runs them concurrently on the input dataset. Each computing unit has a unique ID and is assigned a distinct subset of the input. It has the capability of conducting computations on its data and can only access its local memory that is persistent through the life cycle of the application. The total number of computing units depends on the granularity of their input. To ensure the autonomy of computing units, the user's code must be self-contained, i.e., does not depend on explicit synchronization with other units. As a result, a computing unit may be randomly scheduled on any available resource without affecting the correctness of the user's code.

Communication

This phase allows a computing unit to send fixed-size messages directly to others (point-to-point communication). This stage handles buffering techniques, object serialization, and routing protocols for the messages to ensure a successful delivery to computing units. Messages sent in an iteration are only available to the destination computing units in the next iteration. Moreover, the execution of the communication phase can be overlapped with the computation one, to improve the overall performance of the system, as soon as a computing unit starts sending messages. Several research studies (Salihoglu and Widom 2013; Li et al. 2016; Yan et al. 2015; Goodrich 1999) proposed various techniques to improve the performance of this phase including message aggregation, broadcasting, and utilizing remote



BSP Programming Model, Fig. 1 The BSP compute model

direct memory access (RDMA) to reduce network costs.

Synchronization

The synchronization phase ensures the consistency of the BSP computations and helps to avoid deadlocks. It concludes the end of an iteration by strictly blocking the system from starting the next iteration until all computing units complete their current execution and the outgoing messages reach their destination successfully. It may use one or more synchronization barriers (Khayyat et al. 2013) depending on the nature of the BSP system. For applications with lower consistency requirements (Gonzalez et al. 2012; Xiao et al. 2017), Stale Synchronous Parallel (SSP) (Cui et al. 2014) model may be used to reduce the waiting time for each iteration. This is done by boundingly relaxing the strictness of the synchronization barriers to enable computing units to start the next iteration without waiting for stragglers to finish their current iteration.

Advantages

The BSP model simplifies parallel computations. It allows users to implement parallel applications without worrying about the challenges of parallel environments. The workload of a BSP application can be automatically balanced by increasing the number of computing units. More sophisticated load balancing techniques can also

be used (Bonorden 2007a,b; Khayyat et al. 2013) to ensure a balanced workload of the BSP during runtime. Such explicit load balancing techniques employ the restrictive nature of the BSP model to predict the behavior of the user's code (González et al. 2001) and transparently reassign computing units without affecting the application's correctness. Since computing units are independent, they can be parallelized in shared memory multi-core systems (Valiant 2011; Yzelman et al. 2014), distributed shared-nothing environments (Hill et al. 1998), graphics processing units (GPU) (Hou et al. 2008), parallel system on chip (Bonorden 2008), and heterogeneous clusters (Bonorden 2008).

Limitations

The main disadvantage of BSP is that not all applications can be programmed in its restrictive model. The autonomy and independence of computing units in BSP hinder programmers from developing applications that require high data dependency within its data partitions. Similarly, explicitly synchronizing the state of different computing units may require sophisticated implementation of the user's logic (Sakr et al. 2017; Abdelaziz et al. 2017; Teixeira et al. 2015) or significant modifications to the underlying workflow of the BSP system (Salihoglu and Widom 2014).

BSP also suffers from **strugglers**, i.e., slow computing units as its strict synchronization barriers restrict the execution of the next iteration until all computing units finished and all messages are delivered. The cost of each iteration in BSP is the sum of the execution time of the most delayed computing unit, the cost to transmit or receive the largest messages buffer for a computing unit, and the latency of the synchronization barrier. As a result, the performance of a BSP iteration becomes as fast as its slowest computing unit. Asynchronous iterations or less strict barriers may be used to improve the performance of BSP at the expense of the algorithm's accuracy and convergence rate (Gonzalez et al. 2012). Another way to avoid strugglers is either by excluding expensive computing units from the computation (Abdelaziz et al. 2017), utilizing replication to reduce their load (Salihoglu and Widom 2013; Gonzalez et al. 2012), or shuffling the CPU assignment of expensive computing units (Bonorden 2007a; Khayyat et al. 2013).

Applications

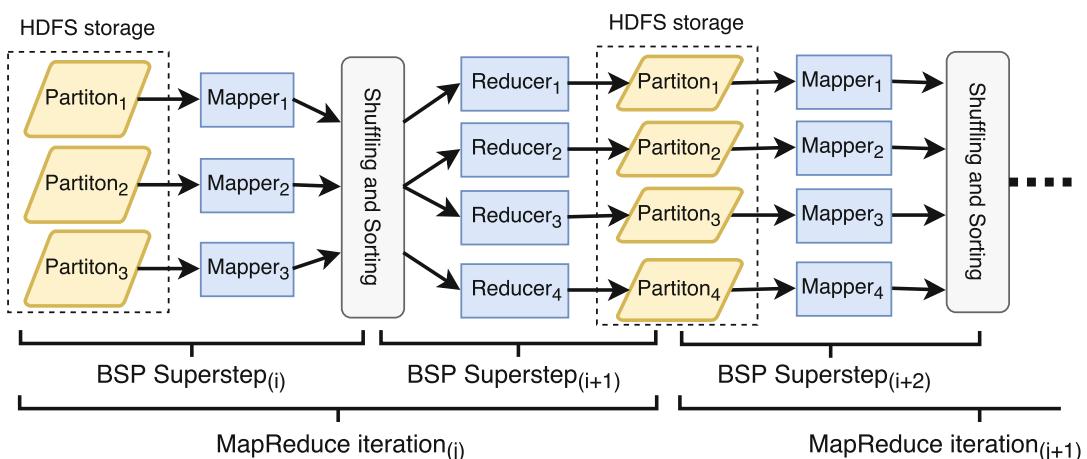
MapReduce

It is a general-purpose distributed system (Dean and Ghemawat 2008) that provides automatic scalability and fault tolerance to programmers through two operators: a **mapper** and a

reducer. Although MapReduce is not originally designed to run as a traditional BSP systems, its execution model follows BSP. That is, mappers and reducers can be considered as computing units of two consecutive supersteps for BSP, while intermediate data shuffling and HDFS I/O play the role of communication and synchronization phases in BSP. As shown in Fig. 2, each MapReduce iteration contains two BSP supersteps. The mapper function is used to generate the distributed computing units of the first superstep ($superstep_{(i)}$), where the execution of the next superstep is held by the shuffling and sorting phase until all mappers finish their execution. The reducer's supertep ($superstep_{(i+1)}$) starts once the shuffling phase completes and the next superstep ($superstep_{(i+2)}$) is blocked until all reducers write their data to HDFS. Alongside mappers and reducers, programmers may also implement a **combiner** that is executed on the output of mappers, i.e., in $superstep_{(i)}$ and $(superstep_{(i+2)})$, to reduce the size of the exchanged messages and thus reduce the network overhead.

Apache Hama

Similar to MapReduce, Apache Hama (Siddique et al. 2016) is also a generalized system that enables automatic parallelization and fault tolerance. It supports a much more comprehensive



BSP Programming Model, Fig. 2 The BSP compute model within MapReduce

range of applications in which it explicitly models its computations on top of the BSP programming model. Apache Hama provides three abstractions to its computing units: general, vertex-based for graph processing, and neuron-based for machine learning. As a result, Apache Hama is able to customize its optimizations to the communication phase and synchronization barriers based on the underlying BSP application.

Pregel

Pregel (Malewicz et al. 2010) and its clones are a typical example of a BSP application that is customized to scale large graph computation distributed systems. Each vertex in the input graph is assigned a distinct compute unit. In each graph iteration, vertices exchange messages about their state and the weights of their graph edges to achieve a newer meaningful state. The communication pattern of a typical Pregel application is usually predictable; a vertex only sends messages to its direct neighbors. As a result, some Pregel clones (Gonzalez et al. 2014) utilized this predictability to improve the overall performance of the BSP's communication phase. Pregel also utilizes aggregators, which is a commutative and associative function, to reduce the size of messages exchanged between the graph vertices. Graph algorithms on Pregel proceed with several BSP iterations until the state of its vertices converge to specific values or reach a maximum predefined limit for the allowed number of iterations.

Cross-References

- ▶ [Graph Processing Frameworks](#)
- ▶ [Parallel Graph Processing](#)

References

- Abdelaziz I, Harbi R, Salihoglu S, Kalnis P (2017) Combining vertex-centric graph processing with SPARQL for large-scale RDF data analytics. *IEEE Trans Parallel Distrib Syst* 28(12):3374–3388
- Beran M (1999) Decomposable bulk synchronous parallel computers. In: Proceedings of the 26th conference on current trends in theory and practice of informatics on theory and practice of informatics, SOFSEM'99. Springer, London, pp 349–359
- Blelloch GE, Gibbons PB, Matias Y, Zagha M (1995) Accounting for memory bank contention and delay in high-bandwidth multiprocessors. In: Proceedings of the seventh annual ACM symposium on parallel algorithms and architectures, SPAA'95. ACM, New York, pp 84–94
- Bonorden O (2007a) Load balancing in the bulk-synchronous-parallel setting using process migrations. In: 2007 IEEE international parallel and distributed processing symposium, pp 1–9
- Bonorden O (2007b) Load balancing in the bulk-synchronous-parallel setting using process migrations. In: Proceedings of the 21st international parallel and distributed processing symposium (IPDPS), Long Beach, pp 1–9, 26–30 Mar 2007. <https://doi.org/10.1109/IPDPS.2007.370330>
- Bonorden O (2008) Versatility of bulk synchronous parallel computing: from the heterogeneous cluster to the system on chip. PhD thesis, University of Paderborn, Germany. <http://ubdok.uni-paderborn.de/servlets/DerivateServlet/Derivate-6787/diss.pdf>
- Cui H, Cipar J, Ho Q, Kim JK, Lee S, Kumar A, Wei J, Dai W, Ganger GR, Gibbons PB, Gibson GA, Xing EP (2014) Exploiting bounded staleness to speed up big data analytics. In: Proceedings of the 2014 USENIX conference on USENIX annual technical conference, USENIX association, USENIX ATC'14, pp 37–48
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Gonzalez JA, Leon C, Piccoli F, Printista M, Roda JL, Rodriguez C, de Sande F (2000) Oblivious BSP. Springer, Berlin/Heidelberg, pp 682–685
- González JA, León C, Piccoli F, Printista M, Roda JL, Rodríguez C, de Sande F (2001) Performance prediction of oblivious BSP programs. Springer, Berlin/Heidelberg, pp 96–105. https://doi.org/10.1007/3-540-44681-8_16
- Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) PowerGraph: distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th USENIX symposium on operating systems design and implementation (OSDI), USENIX, Hollywood, pp 17–30
- Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) GraphX: graph processing in a distributed dataflow framework. In: 11th USENIX symposium on operating systems design and implementation (OSDI). USENIX Association, Broomfield, pp 599–613. <https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez>
- Goodrich MT (1999) Communication-efficient parallel sorting. *SIAM J Comput* 29(2):416–432
- Hill JMD, McColl B, Stefanescu DC, Goudreau MW, Lang K, Rao SB, Suel T, Tsantilas T, Bisseling RH

- (1998) Bsplib: the BSP programming library. *Parallel Comput* 24(14):1947–1980
- Hou Q, Zhou K, Guo B (2008) BSGP: bulk-synchronous GPU programming. In: ACM SIGGRAPH 2008 Papers, SIGGRAPH'08. ACM, New York, pp 19:1–19:12
- Juurlink BHH, Wijshoff HAG (1996) The e-BSP model: incorporating general locality and unbalanced communication into the BSP model. In: Proceedings of the second international Euro-Par conference on parallel processing-volume II, Euro-Par'96. Springer, London, pp 339–347
- Khayyat Z, Awara K, Alonazi A, Jamjoom H, Williams D, Kalnis P (2013) Mizan: a system for dynamic load balancing in large-scale graph processing. In: Proceedings of the 8th ACM European conference on computer systems, EuroSys'13. ACM, pp 169–182
- Li M, Lu X, Hamidouche K, Zhang J, Panda DK (2016) Mizan-RMA: accelerating Mizan graph processing framework with MPI RMA. In: 2016 IEEE 23rd international conference on high performance computing (HiPC), pp 42–51
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, SIGMOD'10. ACM, pp 135–146
- Sakr S, Orakzai FM, Abdelaziz I, Khayyat Z (2017) Large-scale graph processing using apache giraph. Springer International Publishing AG, Cham
- Salihoglu S, Widom J (2013) GPS: a graph processing system. In: Proceedings of the 25th international conference on scientific and statistical database management, SSDBM. ACM, pp 22:1–22:12
- Salihoglu S, Widom J (2014) Optimizing graph algorithms on Pregel-like systems. *Proc VLDB Endow* 7(7):577–588
- Siddique K, Akhtar Z, Yoon EJ, Jeong YS, Dasgupta D, Kim Y (2016) Apache Hama: an emerging bulk synchronous parallel computing framework for big data applications. *IEEE Access* 4:8879–8887
- Teixeira CHC, Fonseca AJ, Serafini M, Siganos G, Zaki MJ, Aboulnaga A (2015) Arabesque: a system for distributed graph mining. In: Proceedings of the 25th symposium on operating systems principles, SOSP'15. ACM, New York, pp 425–440
- Valiant LG (1990) A bridging model for parallel computation. *Commun ACM* 33(8):103–111
- Valiant LG (2011) A bridging model for multi-core computing. *J Comput Syst Sci* 77(1):154–166
- Williams TL, Parsons RJ (2000) The heterogeneous bulk synchronous parallel model. Springer, Berlin/Heidelberg, pp 102–108
- Xiao W, Xue J, Miao Y, Li Z, Chen C, Wu M, Li W, Zhou L (2017) Tux²: distributed graph computation for machine learning. In: 14th USENIX symposium on networked systems design and implementation (NSDI), Boston, pp 669–682
- Yan D, Cheng J, Lu Y, Ng W (2015) Effective techniques for message reduction and load balancing in distributed graph computation. In: Proceedings of the 24th international conference on world wide web, international world wide web conferences steering committee, WWW'15, pp 1307–1317
- Yzelman AN, Bisseling RH, Roose D, Meerbergen K (2014) Multicorebsp for c: a high-performance library for shared-memory parallel programming. *Int J Parallel Prog* 42(4):619–642
-
- ## BSPlib
- ▶ [BSP Programming Model](#)
-
- ## Bulk Synchronous Parallel
- ▶ [BSP Programming Model](#)
-
- ## Business Process Analytics
- Marlon Dumas¹ and Matthias Weidlich²
- ¹ Institute of Computer Science, University of Tartu, Tartu, Estonia
- ² Humboldt-Universität zu Berlin, Department of Computer Science, Berlin, Germany
- ## Synonyms
- [Business process mining](#); [Business process monitoring](#); [Process mining](#); [Workflow mining](#)
- ## Definitions
- Business process analytics is a body of concepts, methods, and tools that allows organizations to understand, analyze, and predict the performance of their business processes and their conformance with respect to normative behaviors, on the basis of data collected during the execution of said processes. Business process analytics encompasses a wide range of methods for extracting, cleansing, and visualizing business process execution data, for discovering business process models

from such data, for analyzing the performance and conformance of business processes both in a postmortem and in an online manner, and finally, for predicting the future state of the business processes based on their past executions.

Overview

This entry provides an overview of the general context of business process analytics and introduces a range of concepts and methods, all the way from the extraction and cleansing of business process event data, to the discovery of process models, the evaluation of the performance and conformance of business processes both offline and online, as well as the management of collections of business process models. These concepts and methods are discussed in detail in subsequent entries of this encyclopedia.

The Context of Business Process Analytics

The context of business process analytics is spanned by a few essential concepts. Below, we start by reviewing the notion of a business process as a means to structure the operations of an organization. We then discuss how the execution of a business process on top of modern enterprise software systems leads to the accumulation of detailed data about the process, which provide the basis for business process analytics. We then discuss the concept of business process model, which plays a central role in the field of business process management in general and business analytics in particular.

Business Processes

The business process management (BPM) discipline targets models and methods to oversee how work is performed in an organization (Dumas et al. 2018). It aims at ensuring consistent outcomes and strives for materializing improvement opportunities. The latter may refer to various operational aspects. Depending on the objectives of an organization, improvement may be defined

as a reduction of costs, execution times, or error rates but may also refer to gaining a competitive advantage through innovation efforts. The essence of BPM, though, is that improvement is not limited to individual steps of an organization's operations. Rather, improvement shall take a holistic view and consider the scope of processes, entire chains of business events, activities, and decisions.

Adopting this view and the definition brought forward by Dumas et al. (2018), a business process is a *collection of inter-related events, activities, and decision points that involve a number of actors and objects, which collectively lead to an outcome that is of value to at least one customer*. Typically, a business process has a well-defined start and end, which gives rise to a notion of an *instance* of the process, also known as *case*. Such a case is the particular occurrence of business events, execution of activities, and realization of decisions to serve a single request by some customer.

The way in which a business process is designed and performed affects both the *quality of service* that customers perceive and the *efficiency* with which services are delivered. An organization can outperform another organization offering similar kinds of service, if it has better processes and executes them better. However, the context of business processes in terms of, for instance, organizational structures, legal regulations, and technological infrastructures, and hence the processes themselves, is subject to continuous change. Consequently, to ensure operational excellence, one-off efforts are not sufficient. Rather, there is a need to continuously assess the operation of a process, to identify bottlenecks, frequent defects and their root causes, and other sources of inefficiencies and improvement opportunities. Business process analytics is a body of techniques to achieve this goal.

Business Process Event Data

Modern companies use dedicated software tools, called *enterprise software systems*, to record, coordinate, and partially conduct everyday work. Specifically, an enterprise software system supports the conduct of a business process by au-

tomating and coordinating business events, activities, and decisions involved in each case of the process. The former means that the reaction to business events, the conduct of activities, or a routine for decision-making may directly be provided by a software system. As an example, one may consider the scenario of a bank processing student loans. Here, a database transaction may be executed to store a loan request after it has been received. This transaction corresponds to an automated activity that is performed immediately in response to the event of receiving the request. Subsequently, a Web service call may be made to check the data provided as part of the request for plausibility, which corresponds to an automated activity that is triggered by reaching a particular state in the respective case.

On the other hand, enterprise software systems coordinate the (potentially manual) efforts of *workers*, people of an organization that are in charge of the conduct of specific activities of the process. In that case, workers see only activities for which they are directly responsible, while the enterprise software system hides the complexity of coordinating the overall process. A worker then faces a number of *work items* that need to be handled. In the above example, a loan officer might see a list of student loan applications that need to be assessed (i.e., accepted or rejected). The assessment of one loan application is a work item, which corresponds to a single activity. However, it also belongs to a specific case, i.e., a specific loan request. A single worker may therefore have multiple work items to handle, all related to the same activity yet belonging to different cases.

An enterprise software system keeps track of the state of each case at any point in time. To this end, it creates a data record, commonly referred to as an *event*, whenever an automated activity is executed, a business event occurs, a worker starts or finishes working on a work item, or a decision is made. Each of these events typically includes additional data, e.g., as it has been provided as a result of executing an activity, whether it was automated (e.g., the response message of a Web service call) or conducted manually (e.g., data inserted by a worker in a Web form). For ex-

ample, after triggering an automated plausibility check of a loan request, an enterprise software system decides whether to forward the request to a worker for detailed inspection. If so, a worker manually determines the conditions under which a loan is offered. A definition of these conditions is then submitted to the system when completing the respective work item.

By coordinating and conducting the activities related to the cases of a business process, an enterprise software system produces an *event stream*. Each event denotes a state change of a case at a particular point in time. Such an event stream may directly be used as input for business process analytics. For instance, using stream processing technology, compliance violations (loans are granted without a required proof of identity of the requester) or performance bottlenecks (final approval of loan offers delays the processing by multiple days) can be detected.

In many cases, however, business process analytics is done in a postmortem manner. In other words, events are recorded and stored in a so-called *event log*, which is analyzed later using a range of techniques. An event log is a collection of events recorded during the execution of (all) cases of a business process during a period of time.

In order to understand what type of analysis we are able to do using an event log, we need to introduce first the concept of business process model. Indeed, a subset of the analysis techniques that can be applied to an event log also involve process models as input, as output, or both. These techniques are commonly known under the umbrella term of *process mining* (van der Aalst 2016).

Business Process Models

A business process model is a representation of a business process, i.e., the way an organization operates to achieve a goal, as outlined above. It represents the activities, business events, and decisions of a process along with their causal and temporal execution dependencies. To this end, a typical business process model is a graph consisting of at least two types of nodes: task nodes and control nodes. Task nodes describe ac-

tivities, whether they are performed by workers or software systems, or a combination thereof. Control nodes along with the structure of the graph capture the execution dependencies, therefore determining which tasks should be performed after completion of a given task. Task nodes may also represent business events that guide the execution of a process, e.g., in terms of delaying the execution until an external trigger has been observed. Decisions, in turn, are commonly represented by dedicated control nodes that define branching conditions for process execution.

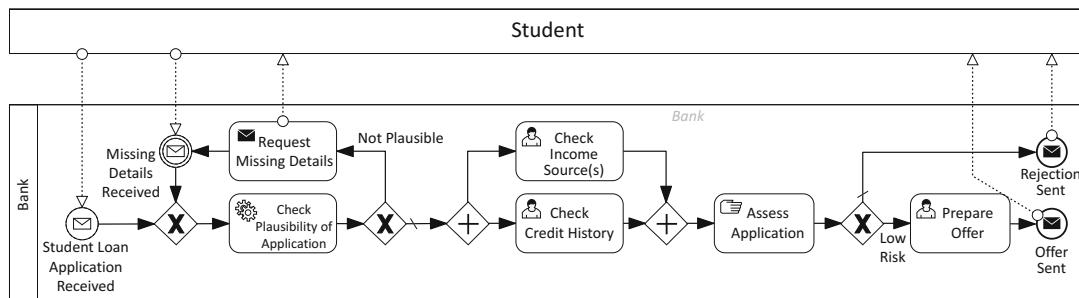
Beyond these essential concepts, process models may capture additional perspectives of a process. They may model the data handled during processing, e.g., by including nodes called data objects that denote inputs and outputs of tasks. Additionally, business process models may include a resource perspective, which captures the roles of involved workers or the types of supporting information systems. In the above example of handling student loan requests, a process model may define the actual application as a data object that is used as input for the activity of checking its plausibility. In addition, the model could specify which specific service provided by which system implements this check.

There are many visual languages that can be used for business process modeling. As an example, Fig. 1 shows a model of the aforementioned student loan process captured in a language called BPMN (Business Process Model and Notation; see also <http://www.bpmn.org/>). An instance of this process, a specific case, is triggered by the

receipt of a student loan application, which is visualized by the leftmost node in the model that represents the respective business event. Subsequently, the application is checked for plausibility, functionality which is provided by an external Web service. Based on the result of this check, the application being plausible or not, a decision is taken. The latter is represented by control node (diamond labeled with “x”). If the application is not plausible, missing details are quested from the applying student, and the plausibility check is conducted again, once these details have been received. The default case of a plausible application (flow arc with a cross bar) leads to two activities being conducted by one or more workers in parallel, i.e., checking the income source(s) and the credit history. This concurrent execution and synchronization are indicated by a pair of control nodes (diamonds labeled with “+”). Subsequently, the application is manually assessed, and a decision is taken, whether to prepare and send an offer or to reject the application.

Business Process Analytics Techniques

Business process analytics comprises a collection of techniques to investigate business processes, based on their representations in the form of event data and/or business process models. Figure 2 provides a general overview of these techniques and the context in which they are defined. This



Business Process Analytics, Fig. 1 A business process model in BPMN that captures the process of applying for a student loan

context is given by a business process as it is supported by an enterprise system, the event data that is recorded during process execution (available as logs or streams), and the models that capture various aspects of a business process.

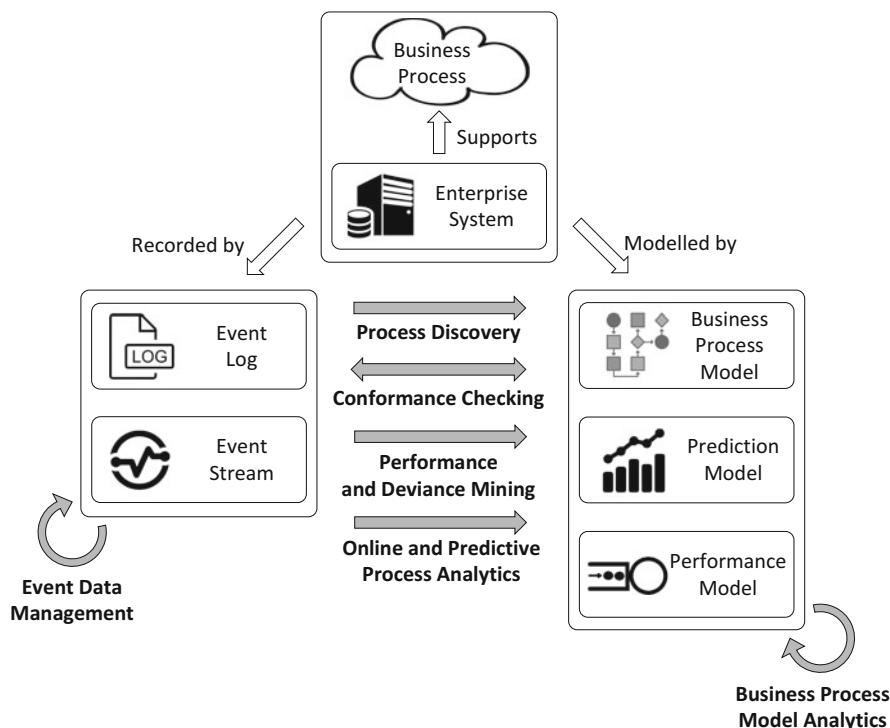
In the remainder of this section, we discuss major categories of techniques for business process analytics techniques indicated in bold in Fig. 2. We further point to other entries in this encyclopedia, which introduce each of these techniques in more detail.

Event Data Management

The left-hand side of Fig. 2 illustrates that the starting point for business process analytics is the availability of event streams and/or event logs generated by the execution of business processes. These event logs can be represented in various ways as will be discussed in the entry *Business Process Event Logs and Visualization*.

The latter entry will also discuss a number of methods for visualizing an event log in order to get a bird's-eye view of the underlying business process.

In their raw form, event streams and event logs suffer from various data quality issues. For example, an event log might be incomplete: some events might be missing altogether, while some events might be missing some attributes (e.g., the time stamp). Due to recording errors, the data carried by some of the events might be incorrect or inconsistent. Also, different events might be defined at different levels of abstraction. For example, some events might correspond to business-level events (e.g., the acceptance of a loan request), while others might be system-level events (e.g., a system-level fault generated by a Web service call to check the plausibility of the data provided as part of a loan request). These data quality issues will also be discussed in the entry on *Business Process Event Logs and Visualization*



Business Process Analytics, Fig. 2 Overview of business process analytics techniques

Visualization, and various techniques to detect and handle them will be presented in the entry on *Event Log Cleaning for Business Process Analytics*.

Process Discovery and Conformance Checking

Assuming that we have a suitably cleaned event log, a first family of business process analytics techniques allows us to either discover a process model from the event stream/log or to compare a given event stream/log to a given process model. These two latter types of techniques are discussed in the entries on *Automated Process Discovery* and *Conformance Checking*, respectively (cf. Fig. 2). Automated process discovery techniques take as input an event log and produce a business process model that closely matches the behavior observed in the event log or implied by the traces in the event log. Meanwhile, conformance checking techniques take as input an event log and a process model and produce as output a list of differences between the process model and the event log. For example, the process model might tell us that after checking the plausibility of a loan application, we must check the credit history. But maybe in the event log, sometimes after the plausibility check, we do not see that the credit history had been assessed. This might be because of an error, or more often than not, it is because of an exception that is not captured in the process model. Some conformance checking techniques take as input an event log and a set of business rules instead of a process model. These techniques check if the event log fulfills the business rules, and if not, they show the violations of these rules.

When comparing an event log and a process model (using conformance checking techniques), various discrepancies might be found between the model and the log. Sometimes these discrepancies are due to data quality issues in the event log, in which case they can be corrected by adjusting the event log, but sometimes it may turn out that the process model is incorrect or incomplete. In this latter case, it might be necessary to adjust (or repair) the process model in order to better fit the event log. Techniques for repairing

a process model with respect to an event log will be discussed in the entry on *Process Model Repair*.

When applied to complex event logs, the output of automated process discovery techniques might be difficult to understand. Oftentimes, this is because the event log captures several variants of the business process rather than just one. For example, an event log of a loan request process might contain requests for various different types of loan and from different types of applicants. If we try to discover a single process model for all these variants, the resulting process model might contain too many execution paths, which makes it difficult to grasp at once. One approach to deal with this complexity is to split the traces in the event log into multiple sub-logs, and to discover one process model for each of the sub-logs. Intuitively, each of these sub-logs contains a set of similar traces and hence represents one possible variant of the process. Techniques for splitting an event log into sub-logs are presented in the entry on *Trace Clustering*.

Trace clustering is only one of several alternative techniques to handle complex event logs. Another approach is to use process modeling notations that are well adapted for capturing processes with high levels of variability. Such approaches are discussed in the entry *Declarative Process Mining*. Another approach is to discover process models with a modular structure. Such approaches are discussed in the entries on *Artifact-Centric Process Mining* and on *Hierarchical Process Discovery*.

Sometimes, event logs are not only complex but also excessively large, to the extent that automated process discovery and conformance checking techniques do not scale up. Divide-and-conquer approaches to deal with such scalability issues are discussed in the entry on *Decomposed Process Discovery and Conformance Checking*.

Finally, closely related to the problem of discovering process models from event logs is that of discovering the business rules that are used in the decision points of the process model. A dedicated entry on *Decision Discovery in Business Processes* deals specifically with this latter problem.

Performance and Deviance Analysis

Another core set of business process analytics techniques are focused on business process performance and deviance analysis (cf. Fig. 2). A prerequisite to be able to analyze process performance is to have a well-defined set of process performance measures. A dedicated entry in the encyclopedia on *Business Process Performance Measurement* deals specifically with this question.

Once we have defined a set of performance measures, it becomes possible to analyze the event log from multiple dimensions (e.g., on a per variant basis, on a per state basis, etc.) and to aggregate the performance of the process with respect to the selected performance measure. A framework for analyzing the performance of business processes with respect to multiple dimensions using so-called *process cubes* is presented in the entry on *Multidimensional Process Analytics*.

Waiting time is a prominent performance measure in the context of business processes, as it plays a role in the perceived quality of a process, and is related to the amount of resources available in a process and hence to the cost of the process. Low waiting times are associated with a leaner and more responsive process. However, low waiting times are not easily achievable when demand is high and when the amount of available resources is limited. When the demand for resources exceeds the available capacity, queues build up, and these queues give rise to waiting times. Understanding the buildup of queues is a recurrent question in business process analytics. Techniques for analyzing queues on the basis of event logs are discussed in the entry on *Queue Mining*.

Naturally, the performance of the process can vary significantly across different groups of cases. For example, it may be that the cycle time 90% of cases of a process is below a given acceptable threshold (e.g., 5 days). However, in some rare cases, the cycle time might considerably exceed this threshold (e.g., 30+ days). *Deviance mining* techniques help us to diagnose the reasons why a subset of the execution of a business process deviate with

respect to the expected outcome or with respect to established performance objectives. Deviance mining techniques are discussed in a dedicated entry of this encyclopedia's section.

Online and Predictive Process Analytics

All the techniques for business process analytics mentioned above are primarily designed to analyze the current “as-is” state of a process in a postmortem setting.

Equally important is the detection of deviations with respect to conformance requirements in real time, as the process unfolds. The entry on *Streaming Process Discovery and Conformance Checking* deals with the question of how automated process discovery and conformance checking can be performed in real time based on a stream of events (as opposed to doing so over an event log collected during a period of time). Such techniques allow us, for example, to detect deviations with respect to the normative behavior (captured by a process model) in real time, thereby enabling immediate implementation of corrective actions.

Sometimes though, it is desirable to be able to preempt undesired situations, such as violations of performance objectives, before these situations happen. This can be achieved by building predictive models from historical event logs and then using these predictive models to make predictions (in real time, based on an event stream) about the future state of running cases of the process. Various techniques to build and validate such predictive models are discussed in the entry on *Predictive Business Process Monitoring*.

When the performance of a business process does not fulfill the objectives of the organization, it becomes necessary to adjust it, for example, by redeployment of resources from some tasks to other tasks of the process, by eliminating or automating certain tasks, or by changing the sequence of execution of the tasks in the process. Business process simulation techniques allow us to evaluate the effects of such changes on the expected performance of the process. In a nutshell, business process simulation means that we take a business process model, enhance it with

simulation parameters (e.g., the rate of creation of process instances, the processing times of tasks in the process, the number of available resources, etc.), and execute it in a simulated environment a large number of times, in order to determine the expected cycle time, cost, or other performance measures. Traditionally, simulation parameters are determined manually by domain experts, based on a sample of cases of a process, or by observing the execution of the process during a certain period of time. However, the availability of event logs makes it possible to build more accurate simulation models by extracting the simulation parameters directly from an event log. The question of how to use event logs for process simulation is discussed in the entry on *Data-Driven Process Simulation*.

Business Process Model Analytics

Business process analytics is not confined to the analysis of event logs. It also encompasses techniques for analyzing collections of process models, irrespective of the availability of data about the execution of a process (Beheshti et al. 2016). We call the latter family of techniques *business process model analytics* techniques (cf., right-hand side of Fig. 2).

The entries on *Business process model matching* and *Business process querying* deal with two families of business process model analytics techniques. The first of these entries deals with the problem of comparing two or more process models in order to characterize their commonalities (or similarities) and their differences. This is a basic operation that allows us, for example, to merge multiple process models into a single one (e.g., to consolidate multiple variants of a process). It also allows us to find process models in a repository that are similar to a given one, which is also known as similarity search. However, business process querying provides a much broader view on how to manage process models. That includes languages to define queries related to particular aspects and properties of processes, along with efficient evaluation algorithms for such queries. As such, business process

querying provides concepts and techniques to make effective use of large collections of process models.

Cross-References

- ▶ [Artifact-Centric Process Mining](#)
- ▶ [Automated Process Discovery](#)
- ▶ [Business Process Deviance Mining](#)
- ▶ [Business Process Event Logs and Visualization](#)
- ▶ [Business Process Model Matching](#)
- ▶ [Business Process Performance Measurement](#)
- ▶ [Business Process Querying](#)
- ▶ [Conformance Checking](#)
- ▶ [Data-Driven Process Simulation](#)
- ▶ [Decision Discovery in Business Processes](#)
- ▶ [Declarative Process Mining](#)
- ▶ [Decomposed Process Discovery and Conformance Checking](#)
- ▶ [Event Log Cleaning for Business Process Analytics](#)
- ▶ [Hierarchical Process Discovery](#)
- ▶ [Multidimensional Process Analytics](#)
- ▶ [Predictive Business Process Monitoring](#)
- ▶ [Process Model Repair](#)
- ▶ [Queue Mining](#)
- ▶ [Streaming Process Discovery and Conformance Checking](#)
- ▶ [Trace Clustering](#)

References

- Beheshti SMR, Benatallah B, Saks S, Grigori D, Nezhad H, Barukh M, Gater A, Ryu S (2016) Process analytics – concepts and techniques for querying and analyzing process data. Springer, Cham
- Dumas M, Rosa ML, Mendling J, Reijers HA (2018) Fundamentals of business process management, 2nd edn. Springer, Berlin
- Wil MP, van der Aalst (2016) Process mining – data science in action, 2nd edn. Springer, Berlin, Heidelberg

Business Process Anomaly Detection

- ▶ [Business Process Deviance Mining](#)

Business Process Audit Trail

- ▶ [Business Process Event Logs and Visualization](#)

Business Process Conformance Checking

- ▶ [Conformance Checking](#)

Business Process Deviance Mining

Francesco Folino and Luigi Pontieri
National Research Council, Institute for High Performance Computing and Networking (ICAR-CNR), Rende (CS), Italy

Synonyms

[Business process anomaly detection](#); [Business process deviation mining](#); [Business process variants analysis](#)

Definitions

Business process deviance mining refers to the problem of (automatically) detecting and explaining deviant executions of a business process based on the historical data stored in a given *Business Process Event Log* (called hereinafter *event log* for the sake of conciseness). In this context, a deviant execution (or “deviance”) is one that deviates from the normal/desirable behavior of the process in terms of performed activities, performance measures, outcomes, or security/compliance aspects.

Usually, the given event log is regarded as a collection of process *traces*, encoding each the history of a single process instance, and the

task amounts to spotting and analyzing the traces that likely represent deviant process (execution) instances.

In principle, this specific process mining task can help recognize, understand, and possibly prevent/reduce the occurrence of undesired behaviors.

Overview

Historically, in a business process mining/intelligence context, the term “deviance mining” was first used in Nguyen et al. (2014). Since then increasing attention has been given to this research topic, owing to two main reasons: (i) deviances may yield severe damages to the organization, e.g., in terms of monetary costs, missed opportunities, or reputation loss; (ii) process logs may be very large and difficult to analyze with traditional auditing approaches, so that automated techniques are needed to discover deviances and/or actionable deviance patterns.

Abstracting from the common idea of exploiting event log data, approaches developed in this field look quite variegated. Indeed, in addition to possibly resorting to different data mining techniques, they may also differ in two fundamental aspects: the analysis *task* and the kinds of *available information*.

Two main deviance mining tasks (often carried out together) have been pursued in the literature: deviance *explanation* and deviance *detection*. The former is devoted to “explain the reasons why a business process deviates from its normal or expected execution” (Nguyen et al. 2014, 2016). In the latter, it must be decided whether a given process instance (a.k.a. *case*) is deviant or not. This task can be accomplished in two fashions: (i) *run-time* detection, each process instance must be analyzed as it unfolds, based on its “premortem” trace; and (ii) *ex post* detection, only “postmortem” traces (i.e., just one fully grown trace for each finished process instance) must be analyzed to possibly discover deviances.

As to the second aspect (namely, available information), two deviance mining settings can be considered, which differ for the presence of auxiliary information, in addition to the given log traces:

- *Supervised*: All the traces are annotated with some deviance label/score, which allows to regard them as examples for learning a deviance detector/classifier or to extract deviance patterns.
- *Unsupervised*: No a priori deviance-oriented information is given for the traces, so that an unsupervised deviance mining method needs to be devised, based on the assumption that all/most deviances look anomalous with respect to the mainstream process behavior.

The rest of this chapter focuses on **ex post** deviance mining approaches. In fact, the exploitation of supervised learning methods to detect deviances at run-time has been considered in the related field of *predictive business process monitoring*.

More specifically, the two following sections discuss supervised and unsupervised deviance mining approaches, respectively. The last section provides some concluding remarks and draws a few directions for future research in this field. All the approaches discussed are summarized in Table 1.

Supervised Deviance Mining

The great majority of supervised deviance mining approaches assume that each log trace is associated with a binary class label telling whether the trace was deemed as deviant or not and try to induce a deviance-oriented classification model for discriminating between these two classes (i.e., deviant vs. normal traces). These approaches are named *classification-based* and described in section “[Classification-Based Deviance Mining](#)”. A few non-classification-based approaches to the discovery of deviance-explanation patterns, from deviance-annotated traces, are then presented in

section “[Other Supervised Deviance Explanation Approaches](#)”.

Classification-Based Deviance Mining

Based on a given collection of example traces, labeled each as either “deviant” or “normal,” these approaches try to induce a deviance-oriented classification model, capable to *detect* whether any new (unlabeled) trace is deviant or not. Moreover, if the model has (or can be easily turned into) a human readable form (e.g., classification rules), it can be used to *explain* the given deviances. In fact, most classification-based approaches tried to pursue both goals (i.e., detection and explanation), by inducing rule/tree-based classifiers.

Three main issues make the induction of a deviance-oriented classification model challenging: (i) process traces are complex sequential data that must be brought to some suitable abstraction level, in order to possibly capture significant discriminative patterns; (ii) obtaining a sufficiently informative training set is difficult, since manually labeling the traces is often difficult/costly and most log traces remain *unlabeled*; and (iii) as in many scenarios, deviant behaviors occur more rarely than normal ones; the training set is often *imbalanced* ([Japkowicz and Stephen 2002](#)).

To cope with the first issue, and reuse standard classifier-induction methods, most approaches adopt a two-phase mining scheme: (1) *log encoding*, the example traces are turned into vectors/tuples in some suitable feature space; and (2) *classifier induction*, the transformed traces (equipped each with a class label) are then made to undergo a propositional classifier-induction algorithm.

In addition to possibly considering global case attributes, most of the encoding schemes adopted in the literature rely on extracting behavioral patterns from the traces and using these patterns as features. As a preliminary step, each event e of a trace is usually abstracted into the associated activity (i.e., the activity executed/referenced to in e), so that the trace is turned into an activity sequence.

The main kinds of patterns used in the literature are as follows ([Nguyen et al. 2016](#)):

- *Individual activities* (IA) (Suriadi et al. 2013): each activity a is regarded as a “singleton” pattern and turned into a numeric feature counting how frequently a occurs in the trace (this corresponds to using a “bag-of-activity” encoding).
- *Activity sets* (SET) (Swinnen et al. 2012): the patterns incorporate subsets of activities that frequently co-occur in the traces (regarded each as an event set, thus abstracting from the execution order).
- *Sequence patterns*: the patterns represent frequent/relevant sequences of activities. The following families of sequence patterns have been considered: *maximal repeats* (MR), *tandem repeats* (TR), *super maximal repeats* (SMR), *near super maximal repeats* (NSMR), the set-oriented (“alphabet”-like) versions of the former (i.e., AMR, ATR, etc.) (Bose and van der Aalst 2013), and (*closed unique*) *iterative patterns* (IP) (Lo et al. 2009) (possibly covering non-contiguous activity subsequences).

In Suriadi et al. (2013), an IA-based log encoding is used to train a decision tree for discriminating and explaining deviances defined as traces with excessively long durations. The discovered rules are then exploited to analyze how the presence of single or multiple activities impacted on deviant behaviors.

A semiautomatic deviation analysis method was proposed in Swinnen et al. (2012), which combines process mining and association rule mining methods, and addresses a scenario where the deviances are cases that deviate from the mainstream behavior, captured by a process model discovered with *automated process discovery* techniques. An association-based classifier is used to characterize deviant cases, based on a set-oriented representation of the traces (which keeps information on both activities and nonstructural case properties).

Bose and van der Aalst (2013) face the discovery of a binary classification model in a scenario where deviances correspond to fraudulent/faulty cases. The traces are encoded propositionally

using one among the following pattern types: TR, MR, SMR, NSMR, the respective *alphabet* versions of the former types, and IA patterns. For the sake of explanation, only association-based classifiers and decision trees are employed. To deal with unlabeled traces, the authors envisaged the possibility to label them automatically in a preprocessing step (with *one-class SVM* and nearest-neighbor methods).

Lo et al. (2009) focus on the case where the deviances are those traces affected by a failure, among those generated by a software process, and introduce the notion of *closed unique iterative patterns* (IPs), as encoding features. These patterns are extracted through an a priori-like scheme and then selected based on their discriminative power (according to *Fischer’s score* Duda et al. 1973). To mitigate the effect of imbalanced classes, a simple oversampling mechanism is exploited.

Nguyen et al. (2014, 2016) define a general scheme uniforming previous pattern-based deviance mining works, which consists of the following steps (to be precise, steps 1 and 3 are only considered in Nguyen et al. 2016): (1) oversampling (to deal with class imbalance), (2) pattern mining, (3) pattern selection (based on Fischer’s score), (4) pattern-based log encoding, and (5) classifier induction. As to the former three steps, the authors used one among the pattern types described above (namely, IA, SET, TR, MR, ATR, AMR, and IP) in isolation. In addition, a few heterogeneous encodings are explored in Nguyen et al. (2014), mixing IA patterns with one of the other *composite* patterns (i.e., SET, TR, MR, ATR, AMR, and IP). Three standard induction methods are used to implement the last step: the decision-tree learning algorithm J48, a nearest-neighbor procedure, and a neural-network classifier. Different instantiations of this scheme (obtained by varying the kind of patterns and the induction method) are tested in Nguyen et al. (2016) on several real datasets.

To assess the explanatory power of the models discovered, the authors evaluate the rulesets obtained by applying J48 to each of the pattern-based encodings. Rulesets discovered with composite pattern types (especially IP,

MRA, and SET) looked more interesting than IA-based ones. As to the capability to correctly classify new test traces, composite patterns do not seem to lead to substantial gain when the process analyzed is lowly structured and/or the deviance is defined in terms of case duration. Composite patterns conversely looked beneficial on the logs of structured processes, but there is no evidence for bestowing a pattern type as the winner.

In fact, the predictiveness of different pattern types may well depend on the application domain. In principle, even on the same dataset, a pattern type might be better at detecting a certain kind of deviance than another type and vice versa. On the other hand, even selecting the classifier-induction method is not that simple, if one relaxes the requirement that the classification model must be self-explaining. Indeed, higher-accuracy results can be achieved by using more powerful classifiers than those considered in the approaches described above. These observations explain the recent surge of multi-encoding hybrid methods, like those described next.

An ensemble-based deviance mining approach is proposed in Cuzzocrea et al. (2015, 2016b) that automatically discovers and combines different kinds of patterns (namely, the same as in Nguyen et al. 2014) and different classifier-induction algorithms. Specifically, different base classifiers are obtained by training different algorithms against different pattern-based encodings of the log. The discovered base classifiers are then combined through a (context-aware) stacking procedure. An oversampling mechanism is applied prior to pattern extraction, in order to deal with class imbalance.

The approach is parametric to the pattern types and learners employed and can be extended to incorporate other encoding/learning methods. In fact, a simplified version of this approach was combined in Cuzzocrea et al. (2016a) with a conceptual event clustering method (to automatically extract payload-aware event abstractions).

A simple semi-supervised learning scheme is used in Cuzzocrea et al. (2016b) to acquire some feedback from the analyst, on the classifications

made by the discovered model on new unlabeled traces, and eventually improve the model.

Other Supervised Deviance Explanation Approaches

Fani Sani et al. (2017) propose to extract explanations from labeled (deviant vs. normal) traces by applying classic subgroup discovery techniques (Atzmueller 2015), instead of rule-based classifier-induction methods, after encoding all the traces propositionally. As a result, a number of interesting logical patterns (namely, conjunctive propositional formulas) are obtained, identifying each a trace “subgroup” that looks unusual enough w.r.t. the entire log (in terms of class distribution).

Folino et al. (2017a) consider instead a setting where interesting self-explaining deviance scenarios are to be extracted from training traces associated with numerical deviance indicators (e.g., concerning performance/conformance/risk measures). The problem is stated as the discovery of a deviance-aware conceptual clustering, where each cluster needs to be defined via descriptive (not a priori known) classification rules – specifically, conjunctive formulas over (both structural and nonstructural) trace attributes and context factors.

Unsupervised Deviance Mining

Unsupervised deviance mining approaches can be divided in two main categories (Li and van der Aalst 2017), namely, *model-based* and *clustering/similarity-based*, which are discussed next in two separate subsections.

Model-Based Approaches

Since, in general, there may be no a priori process model that completely and precisely defines deviance/normality behaviors, model-based approaches rely on inducing mining such a model from the given traces and then classifying as deviant all those that do not comply enough with the model. These approaches are next described in separate paragraphs, according to the kind of behavioral model adopted.

Approaches using a workflow-oriented model

Most model-based approaches employ a control-flow model, combined with conformance checking techniques (van der Aalst et al. 2005). Earlier approaches (e.g., Bezerra et al. 2009) simply mined the model by directly applying traditional control-flow-oriented techniques developed in the related field of *automated process discovery*. However, as generally the log is not guaranteed to only contain normal behavior, the effectiveness of such an approach suffers from the presence of outliers. To better deal with outliers, one could think of applying methods for automatically filtering out infrequent behavior, like the one proposed in Conforti et al. (2017). In particular, in Conforti et al. (2017), first an automaton is induced from the log, in order to approximately model the process behavior, in terms of direct follow dependencies; infrequent transitions are then removed from the automaton; in a final replay of the traces, any behavior that cannot be mapped on the automaton is deemed as an outlying feature.

Bezerra and Wainer (2013) propose three ad hoc algorithms to detect deviant traces in a log L , named “threshold,” “iterative,” and “sampling” algorithm, respectively. All the algorithms rely on comparing each trace t with a model M_t induced from a subset of L (namely, $L - \{t\}$ in the threshold and iterative algorithms and a random sample of L in the sampling algorithm). The anomalousness of t is estimated by either checking if t is an instance of M_t or considering the changes needed to allow M_t to represent t . The last algorithm achieved the best results.

All the works above focused on control-flow aspects only while disregarding other perspectives (e.g., timestamps and resources/executors). Some efforts have been spent recently to detect anomalies pertaining such nonfunctional aspects.

In the two-phase multi-perspective approach of Böhmer and Rinderle-Ma (2016), a (“basic likelihood”) graph encoding the likelihood of a transition between two activities is first induced from the log, assuming that the latter only stores normal behaviors. This graph is then extended by inserting nodes representing resources and ab-

stracted timestamps (extracted from log events), to model the execution of each activity in a more refined way. The resulting graph G is used to assess whether a new trace t is anomalous or not, possibly at run-time. Precisely, the likelihood of t w.r.t. G is compared to the lowest one of the (presumed normal) traces in the training log: if the former is lower than the latter, the trace is marked as anomalous/deviant.

The work in Rogge-Solti and Kasneci (2014) is meant to recognize temporal anomalies concerning activity durations – precisely, cases where a group of interdependent activities shows an anomalous run-time. To detect such anomalies, a stochastic Petri net (Rogge-Solti et al. 2013) is exploited, which can capture statistical information for both decision points (modeled as *immediate transitions*) and the duration of activities (modeled as *timed transitions*). Any new trace is confronted to this model (in terms of log-likelihood) to decide whether it is deviant or not.

Approaches using non-workflow-oriented models

Anomalies concerning the sequence of activities in a trace are detected in Linn and Werth (2016) by reusing variable order Markov models: a sequence is anomalous if at least one activity of the sequence is predicted with a probability value lower than a certain threshold.

A one-class SVM strategy is exploited in Quan and Tian (2009), after converting the contents of the given log into vectorial data. This way, a hypersphere is determined for the training set, assuming that normal behaviors are much more frequent than deviant ones. Any (possibly unseen) data instance lying outside this hypersphere is eventually classified as deviant.

A multi-perspective approach has been proposed in Nolle et al. (2016), which hinges on the induction of an autoencoder-based neural network. The network is meant to play as a (black box) general-enough behavioral model for the log, trained to reproduce each trace step by step. If an input trace and the corresponding reproduced version are not similar enough (w.r.t. a manually defined threshold), the former is classified as deviant. The capacity of the auto-encoding

layers is kept limited, in order to make the network robust to both outliers and noise.

Similarity/Clustering-Based Approaches

These approaches rely on the assumption that normal traces occur in dense neighborhoods or in big enough clusters, rather than on using behavioral models (for normal/deviant traces). Consequently, a trace that is not similar enough to a consistent number of other traces is deemed as deviant. This requires some suitable similarity/distance/density function and/or clustering algorithm to be defined for the traces – see the *Trace Clustering* entry for details on this topic.

Similarity-based approaches In general, kNN-based methods (where NN stands for nearest neighbor) detect outliers by comparing each object with its neighbors, based on a distance measure. In Hsu et al. (2017) the distance measure is defined in terms of activity-level durations and contextual information. More specifically, contextual information is represented in a fuzzy way, and associated membership functions are used to adjust activity-level durations. Distances are then calculated using the differences between adjusted durations, and a kNN scheme is eventually used to identify anomalous process instances.

A local outlier factor (LOF) method is adopted instead in Kang et al. (2012) to spot process instances with abnormal terminations. Notably, the method can be applied to future enactments of the process.

A profile-based deviation detection approach is proposed in Li and van der Aalst (2017), which relies on deeming as deviant any case that is not similar enough to a set CS of mainstream cases in the log. Specifically, a preliminary version of CS is first sampled according to a (dynamically modified) “norm” function, biased toward “more normal” cases. Then, the similarity between each log case and CS is computed, by using some suitably defined “profile” function – in principle, this function can be defined as to capture different perspectives, but the authors only show the application of two alternative control-flow-oriented profile functions. As a result, the log is

partitioned into two sets: normal cases (CN) and deviating cases (CD). Then, the norm function is adjusted as to increase (resp., decrease) the likelihood that normal (resp., deviating) cases are sampled. These two steps (i.e., sampling and profile-based classification) can be iterated, in order to improve the partitioning scheme, before eventually returning the deviant cases.

Clustering-based approaches As noticed in Li and van der Aalst (2017), model-based approaches (especially those hinging on control-flow models) work well on regular and structured processes but have problems in dealing with less structured ones exhibiting a variety of heterogenous behaviors.

By contrast, clustering-based anomaly/outlier detection approaches assume that normal instances tend to form large (and/or dense) enough clusters, and anomalous instances are those belonging to none of these clusters.

In Ghionna et al. (2008) and Folino et al. (2011), log traces are first clustered in a way that the clusters reflect normal process execution variants, so that a case is labeled as deviant if it belongs either to a very small cluster (w.r.t. a given cardinality threshold) or to no cluster at all. To capture normal execution schemes, a special kind of frequent structural (i.e., control-flow oriented) patterns is discovered; the traces are then clustered based on their correlations with these patterns – specifically, the clustering procedure follows a co-clustering scheme, focusing on the associations between traces and patterns, rather than using a pattern-based encoding of the traces. To possibly unveil links between the discovered structural clusters and nonstructural features of the process, a decision tree is discovered (using an ad hoc induction algorithm), which can be used to classify novel process instances at run-time. In principle, the clustering procedure could be performed by using other methods (cf. *Trace Clustering*). However, as noticed in Folino et al. (2011), one should avoid algorithms, like k-means, that are too sensitive to noise/outliers, and may hence fail to recognize adequately the groups of normal instances.

Hybrid solutions All the unsupervised approaches described so far focus on the detection of deviances, without paying attention to their explanation. Clearly, one could perform ex post explanation tasks by simply resorting to supervised classification techniques, after having annotated deviant and normal traces with different class labels, as done in Swinnen et al. (2012). For approaches returning a numerical anomaly score, one could also think of directly applying a rule-based regression method or a conceptual clustering method like that proposed in Folino et al. (2017a) (cf. section “[Other Supervised Deviance Explanation Approaches](#)”).

A hybrid discovery approach (both model-based and clustering-based) was defined in Folino et al. (2017b), which synergistically pursues both detection and explanation tasks. Given a set of traces, associated each with stepwise performance measurements, a new kind of performance-aware process model is searched for, consisting of the following components: (i) a list of easily interpretable classification rules defining distinguished “deviance” scenarios for the process that generated the traces; (ii) a state-based performance model for each discovered scenario and one for the “normal” executions belonging to no deviance scenario. The discovery problem is stated as that of optimizing the model of normal process instances while ensuring that all the discovered performance models are both general and readable enough. The approach mainly exploits a conceptual clustering procedure (addressed to greedily spot groups of traces that maximally deviate from the current normality model M_0), embedded in an greedy optimization scheme (which iteratively tries to improve the normality model M_0 , initially learnt from the entire log).

Discussion and Directions of Future Research

For the sake of summarization and of immediate comparison, some major features of the deviance

mining techniques discussed so far are reported in Table 1.

Notably, these techniques can turn precious in many application scenarios, such as those concerning the detection/analysis of software systems’ failures (Lo et al. 2009), frauds (Yang and Hwang 2006), network intrusions (Jalali and Baraani 2012), SLA violations, etc.

In general, in dynamical application scenarios, supervised techniques should be extended with incremental/continuous learning mechanisms, in order to deal with concept drift issues. In particular, in security-oriented scenarios where deviances correspond to attacks of ever-changing forms, the risk that the classifier induced with supervised techniques is obsolete/incomplete may be very high.

In such a situation, unsupervised deviance mining techniques could be a valuable solution, especially when the analyst is mainly interested in curbing the number of false negatives (i.e., undetected deviances). Among unsupervised deviance detection approaches, those based on clustering/similarity methods are likely to fit better lowly structured business processes (Li and van der Aalst 2017) than those relying on a (possibly extended) control-flow model.

The opportunity to use unlabeled data for classification-based deviance mining has been given little attention so far (Cuzzocrea et al. 2016b). Extending current approaches with semi-supervised learning methods (Wang and Zhou 2010), to exploit such information, is an interesting direction for future research. On the other hand, cost-sensitive learning methods (Elkan 2001) could help deal with imbalanced data more flexibly and allow to make the learner pay more attention to one kind of misclassification (i.e., either false deviances or missed deviances).

Finally, it seems valuable to extend outlier-explanation methods (Angiulli et al. 2009) to extract readable explanations for the deviances found with unsupervised deviance mining techniques.

Business Process Deviance Mining, Table 1 Summary of the approaches described here

Setting	Task	Strategy	Reference
Supervised (label-driven)	Detection, explanation	Classifier induction (IA patterns)	Suriadi et al. (2013)
	Detection, explanation	Classifier induction (SET patterns)	Swinnen et al. (2012)
	Detection, explanation	Classifier induction (non IP sequence patterns)	Bose and van der Aalst (2013)
	Detection, explanation	Classifier induction (iterative patterns (IPs))	Lo et al. (2009)
	Detection, explanation	Classifier induction (IA and sequence patterns)	Nguyen et al. (2014)
	Detection, explanation	Classifier induction (IA and sequence patterns)	Nguyen et al. (2016)
	Detection	Classifier ensemble induction (IA and sequence patterns)	Cuzzocrea et al. (2015)
	Detection	Classifier ensemble induction (IA and sequence patterns)	Cuzzocrea et al. (2016b)
	Detection	Classifier ensemble induction (“abstract” IA and sequence patterns)	Cuzzocrea et al. (2016a)
	Explanation	Subgroup discovery	Fani Sani et al. (2017)
Unsupervised (anomaly-driven)	Explanation	Deviance-oriented conceptual clustering	Folino et al. (2017a)
	Detection	Model-based (pure control-flow model)	Bezerra et al. (2009)
	Detection	Model-based (pure control-flow model)	Bezerra and Wainer (2013)
	Detection	Model-based (time-aware control-flow model)	Rogge-Solti and Kasneci (2014)
	Detection	Model-based (multi-perspective process model)	Böhmer and Rinderle-Ma (2016)
	Detection	Model-based (not process-oriented VOMM)	Linn and Werth (2016)
	Detection	Model-based (notprocess-oriented 1-SVM)	Quan and Tian (2009)
	Detection	Model-based (not process-oriented autoencoder)	Nolle et al. (2016)
	Detection	Similarity-based (k-NN)	Hsu et al. (2017)
	Detection	Similarity-based (LOF)	Kang et al. (2012)
	Detection	Similarity-based (“profile-biased” sampling)	Li and van der Aalst (2017)
	Detection	Clustering-based (pattern-aware co-clustering)	Ghionna et al. (2008)
	Detection	Clustering-based (pattern-aware co-clustering)	Folino et al. (2011)
	Detection, explanation	Hybrid solution (performance-aware process models + conceptual clustering)	Folino et al. (2017b)

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Business Process Event Logs and Visualization](#)
- ▶ [Predictive Business Process Monitoring](#)
- ▶ [Trace Clustering](#)

References

- Angiulli F, Fassetti F, Palopoli L (2009) Detecting outlying properties of exceptional objects. *ACM Trans Database Syst (TODS)* 34(1):7
- Atzmueller M (2015) Subgroup discovery – advanced review. *Wiley Int Rev Data Min Knowl Disc* 5(1):35–49
- Bezerra F, Wainer J (2013) Algorithms for anomaly detection of traces in logs of process aware information systems. *Inf Syst* 38(1):33–44
- Bezerra F, Wainer J, van der Aalst WM (2009) Anomaly detection using process mining. *Enterp Bus Process Inf Syst Model* 29:149–161
- Böhmer K, Rinderle-Ma S (2016) Multi-perspective anomaly detection in business process execution events. In: On the move to meaningful internet systems (OTM'16), pp 80–98
- Bose RJC, van der Aalst WM (2013) Discovering signature patterns from event logs. In: 2013 IEEE symposium on computational intelligence and data mining (CIDM). IEEE, pp 111–118
- Conforti R, La Rosa M, ter Hofstede AH (2017) Filtering out infrequent behavior from business process event logs. *IEEE Trans Knowl Data Eng* 29(2):300–314
- Cuzzocrea A, Folino F, Guarascio M, Pontieri L (2015) A multi-view learning approach to the discovery of deviant process instances. In: OTM confederated international conference on the move to meaningful internet systems. Springer, pp 146–165
- Cuzzocrea A, Folino F, Guarascio M, Pontieri L (2016a) A multi-view multi-dimensional ensemble learning approach to mining business process deviances. In: 2016 international joint conference on neural networks (IJCNN). IEEE, pp 3809–3816
- Cuzzocrea A, Folino F, Guarascio M, Pontieri L (2016b) A robust and versatile multi-view learning framework for the detection of deviant business process instances. *Int J Coop Inf Syst* 25(04):1740003
- Duda RO, Hart PE, Stork DG (1973) Pattern classification. Wiley, New York
- Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of 17th international joint conference on artificial intelligence (IJCAI'01), pp 973–978
- Fani Sani M, van der Aalst W, Bolt A, García-Algarra J (2017) Subgroup discovery in process mining. In: Proceedings of the 20th international conference on business information systems, BIS 2017, Poznan, 28–30 June 2017, vol 288. Springer, p 237
- Folino F, Greco G, Guzzo A, Pontieri L (2011) Mining usage scenarios in business processes: outlier-aware discovery and run-time prediction. *Data Knowl Eng* 70(12):1005–1029
- Folino F, Guarascio M, Pontieri L (2017a) A descriptive clustering approach to the analysis of quantitative business-process deviances. In: Proceedings of the 32nd ACM SIGAPP symposium on applied computing (SAC'17). ACM, pp 765–770
- Folino F, Guarascio M, Pontieri L (2017b) Deviance-aware discovery of high quality process models. In: Proceedings of the 29th IEEE international conference on tools with artificial intelligence (ICTAI'17), pp 724–731
- Ghionna L, Greco G, Guzzo A, Pontieri L (2008) Outlier detection techniques for process mining applications. *Lect Notes Comput Sci* 4994:150–159
- Hsu P-Y, Chuang Y-C, Lo Y-C, He S-C (2017) Using contextualized activity-level duration to discover irregular process instances in business operations. *Inf Sci* 391:80–98
- Jalali H, Baraani A (2012) Process aware host-based intrusion detection model. *Int J Commun Netw Inf Secur* 4(2):117
- Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. *Intell Data Anal* 6(5):429–449
- Kang B, Kim D, Kang S-H (2012) Real-time business process monitoring method for prediction of abnormal termination using KNNI-based LOF prediction. *Exp Syst Appl* 39(5):6061–6068
- Li G, van der Aalst WM (2017) A framework for detecting deviations in complex event logs. *Intell Data Anal* 21(4):759–779
- Linn C, Werth D (2016) Sequential anomaly detection techniques in business processes. In: Proceedings of international conference on business information systems. Springer, pp 196–208
- Lo D, Cheng H, Han J, Khoo S-C, Sun C (2009) Classification of software behaviors for failure detection: a discriminative pattern mining approach. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 557–566
- Nguyen H, Dumas M, La Rosa M, Maggi FM, Suriadi S (2014) Mining business process deviance: a quest for accuracy. In: OTM confederated international conference on the move to meaningful internet Systems. Springer, pp 436–445
- Nguyen H, Dumas M, La Rosa M, Maggi FM, Suriadi S (2016) Business process deviance mining: review and evaluation. arXiv preprint, arXiv:1608.08252
- Nolle T, Seeliger A, Mühlhäuser M (2016) Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In: International conference on discovery science. Springer, pp 442–456
- Quan L, Tian G-s (2009) Outlier detection of business process based on support vector data description. In: ISECS international colloquium on computing, communication, control, and management, CCCM 2009, vol 2. IEEE, pp 571–574

- Rogge-Solti A, Kasneci G (2014) Temporal anomaly detection in business processes. In: International conference on business process management. Springer, pp 234–249
- Rogge-Solti A, van der Aalst WM, Weske M (2013) Discovering stochastic petri nets with arbitrary delay distributions from event logs. In: Proceedings of international conference on business process management. Springer, pp 15–27
- Suriadi S, Wynn MT, Ouyang C, ter Hofstede AH, van Dijk NJ (2013) Understanding process behaviours in a large insurance company in australia: a case study. In: International conference on advanced information systems engineering. Springer, pp 449–464
- Swinnen J, Depaire B, Jans MJ, Vanhoof K (2012) A process deviation analysis – a case study. In: Proceedings of international conference on business process management, pp 87–98
- van der Aalst WMP, De Beer HT, van Dongen BF (2005) Process mining and verification of properties: an approach based on temporal logic. In: Proceedings confederated international conference on the move to meaningful Internet systems: CoopIS, DOA, and ODBASE, pp 130–147
- Wang W, Zhou Z-H (2010) A new analysis of co-training. In: Proceedings of the 27th international conference on machine learning (ICML), pp 1135–1142
- Yang W-S, Hwang S-Y (2006) A process-mining framework for the detection of healthcare fraud and abuse. *Exp Syst Appl* 31(1):56–68

Business Process Deviation Mining

► [Business Process Deviance Mining](#)

Business Process Event Logs and Visualization

Marlon Dumas¹ and Jan Mendling²

¹Institute of Computer Science, University of Tartu, Tartu, Estonia

²WU Vienna, Vienna, Austria

Synonyms

[Business process audit trail](#); [Business process execution log](#); [Business process execution trail](#); [Workflow audit trail](#); [Workflow log](#)

Definitions

In the context of business process analytics, an *event log* is a collection of time-stamped event records produced by the execution of a business process. Each event record tells us something about the execution of a work item (a task) of the process. For example, an event record may tell us that a given task started or completed at a given point in time, or it tells us that a given message event, escalation event, or other relevant event has occurred in a given case in the process.

Overview

The execution of business processes is often supported by software systems. These systems can support isolated tasks, facilitate coordination, or record observations from sensors. All these systems have in common is that they store event data. These events bear valuable information that informs us about how a business process was executed and about its performance. However, such event data is often not directly available in a format that it can be used for business process analytics. For this reason, we typically have to create an event log from the event data that is stored in various systems.

An *event log* is a collection of time-stamped event records. Each event record tells us something about the execution of a work item (and hence a task) of the process (e.g., that a task has started or has been completed), or it tells us that a given message event, escalation event, or other relevant event has occurred in the context of a given case in the process. For example, an event record in an event log may capture the fact that Chuck has confirmed a given purchase order at a given point in time.

Figure 1 illustrates what data is typically stored in an event log. We can see that a single event has a unique event ID. Furthermore, it refers to one individual case, it has a timestamp, and it shows which resources executed which task. These may be participants (e.g., Chuck and Susi) or software systems (SYS1, SYS2, DMS). For most techniques discussed in this chapter, it

is a minimum requirement that for each event in the log, we have three attributes (i) in which case the event occurred (*case identifier*), (ii) which task the event refers to (herein called the *event class*), and (iii) when the event occurred (i.e., a *timestamp*). In practice, there may be other event attributes, such as the resource who performed a given task, the cost, or domain-specific data such as the loan amount in the case of a loan origination process.

Business process event logs such as the above one can be processed and analyzed using process mining tools (cf. entry on Business Process Analytics). These tools provide various approaches for visualizing event logs in order to understand their performance, in particularly, with respect to time-related performance measures.

In the rest of this entry, we discuss how event logs are usually represented (file formats), and we then discuss different approaches for visualizing event logs.

The XES Standard

The event log of Fig. 1 is captured as a list in a tabular format. Simple event logs such as this one are commonly represented as tables and stored in a comma-separated values (CSV) format. This representation is suitable when all the events in the log have the same set of attributes. However, in event logs where different events have different sets of attributes, a flat CSV file is a less suitable representation. For example, in an event log of a loan application process, some events may have customer-related attributes (e.g., age of the customer, income, etc.), while other events contain attributes about the loan application assessment (e.g., the credit risk score). If we represented this event log in a CSV file, we would have to use (null) values to denote the fact that some attributes are not relevant for a given event. Also, in CSV files, the types of the attributes are not explicitly represented, which can be error-prone, for example, when dealing with dates, spatial coordinates, etc.

A more versatile file format for storing and exchanging event logs is the *extensible*

event stream (XES) format (<http://www.xes-standard.org>) standardized by the *IEEE Task Force on Process Mining* and published as IEEE 1849 (Acampora et al. 2017) (<http://www.win.tue.nl/ieeetfpm>). The majority of process mining tools can handle event logs in XES. The structure of an XES file is based on a data model, partially depicted in Fig. 2. An XES file represents an event log. It contains multiple traces, and each trace can contain multiple events. All of them can contain different attributes. An attribute has to be either a string, date, int, float, or Boolean element as a key-value pair. Attributes have to refer to a global definition. There are two global elements in the XES file: one for defining trace attributes and the other for defining event attributes. Several classifiers can be defined in XES. A classifier maps one of more attributes of an event to a label that is used in the output of a process mining tool. In this way, for instance, events can be associated with tasks.

Figure 3 shows an extract of an XES log corresponding to the event log in Fig. 1. The first global element (scope = “trace”) in this log tells us that every trace element should have a child element called “concept:name.” This sub-element will be used to store the case identifier. In this example, there is one trace that has a value of 1 for this sub-element. At the level of individual events, there are three expected elements (i.e., “global” element with scope = “event”): “concept:name,” “time:timestamp,” and “resource.” The “concept:name” sub-element of an event will be used to store the name of the task to which the event refers. The “time:timestamp” and “resource” attributes give us the time of occurrence of the event and who performed it, respectively. The rest of the sample XES log represents one trace with two events. The first one refers to “check stock availability,” which was completed by SYS1 on the 30 July 2012 at 11:14. The second event captures “retrieve product from warehouse” conducted by Rick at 14:20.

The XES standard also supports an attribute “lifecycle:transition” that is meant to describe the transition of a task from one execution state to another. An extensive table of predefined transitions is included in the standard. The most prominent

Case ID	Event ID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve product from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1
3	Ma-6694444444-3	2012-08-02 16:04	Manufacture product	SYS1
3	Co-6751157407-3	2012-08-02 16:12	Confirm order	Chuck
3	Em-6895833333-3	2012-08-02 16:33	Emit invoice	SYS2
3	Sh-7013888889-3	2012-08-02 16:50	Get shipping address	SYS2
3	Ge-7069444444-3	2012-08-02 16:58	Ship product	Emil
3	Re-4305555556-3	2012-08-06 10:20	Receive payment	SYS2
3	Ar-4340277778-3	2012-08-06 10:25	Archive order	DMS
4	Ch-3409722222-4	2012-08-04 08:11	Check stock availability	SYS1
4	Re-5000115741-4	2012-08-04 12:00	Retrieve product from warehouse	SYS1
4	Co-5041898148-4	2012-08-04 12:06	Confirm order	Hans
4	Ge-5223148148-4	2012-08-04 12:32	Get shipping address	SYS2
4	Em-4034837963-4	2012-08-08 09:41	Emit invoice	SYS2
4	Re-4180555556-4	2012-08-08 10:02	Receive payment	SYS2
4	Sh-5715277778-4	2012-08-08 13:43	Ship product	Susi
4	Ar-5888888889-4	2012-08-08 14:08	Archive order	DMS

Business Process Event Logs and Visualization, Fig. 1 Example of an event log for the order-to-cash process

transitions are “assign” to indicate that a task has been assigned to a resource; “start” to record that work on the task has started; “complete” signalling that a task is done; “ate_abort” to indicate that a task was canceled, closed, or aborted; and “pi_abort” to indicate that a whole process instance is aborted.

Challenges of Extracting Event Logs

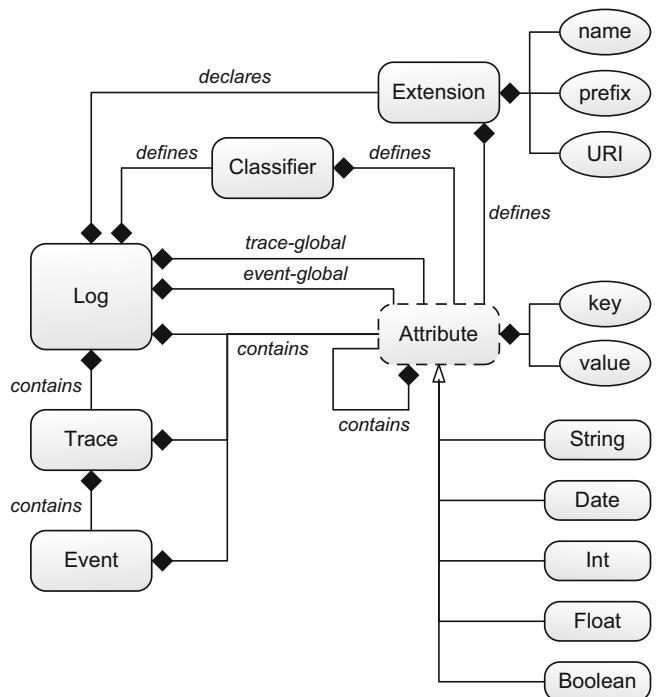
Event data that is available in tabular format as in Fig. 1 can be readily converted to XES. In many cases though, the data which is relevant for event logs is not directly accessible in the required for-

mat but has to be extracted from different sources and integrated. This extraction and integration effort is generally not trivial. A list of 27 potential issues is described in Suriadi et al. (2017). We focus here on four major challenges for log data extraction:

Correlation Challenge

This refers to the problem of identifying the case an event belongs to. Many enterprise systems do not have an explicit notion of process defined. Therefore, we have to investigate which attribute of process-related entities might serve as a case identifier. Often, it is possible to utilize entity identifiers such as order number, invoice number,

Business Process Event Logs and Visualization,
Fig. 2 Metamodel of the XES format



or shipment number. Approaches to infer missing case identifiers are presented in Bayomie et al. (2015) and Pourmirza et al. (2017).

Timestamping Challenge

The challenge to work properly with timestamps stems from the fact that many systems do not consider logging as a primary task. This means that logging is often delayed until the system has idle time or little load. Therefore, we might find sequential events with the same timestamp in the log. This problem is worsened when logs from different systems potentially operating in different time zones have to be integrated. Partially, such problems can be resolved with domain knowledge, for example, when events are known to always occur in a specific order. An approach to repair logs with incomplete or inaccurate timestamps is presented in Rogge-Solti et al. (2013). For estimating the exact timestamp, we can use knowledge about the typical order of tasks and their typical temporal distance.

Longevity Challenge

For long running processes (e.g., processes with cycle times in the order of weeks or months),

we might not yet have observed all cases that started during a recent time period (e.g., cases that started in the past 6 months) up to their completion. In other words, some cases might still be incomplete. Mixing these incomplete cases with completed cases can lead to a distorted picture of the process. For example, the cycle times or other performance measures calculated over incomplete cases should not be mixed with those of the completed ones, as they have different meaning. In order to avoid such distortions, we should consider the option of excluding incomplete cases from an event log. Also, we should ensure that the time period covered by an event log is significantly longer than the mean cycle time of the process so that the event log contains samples of both short-lived and long-lived cases.

Granularity Challenge

Typically, we are interested in conducting event log analysis on a conceptual level for which we have process models defined. In general, the granularity of event log recording is much finer such that each task of a process model might map to a set of events. For example, a task like “retrieve

Business Process Event

Logs and Visualization,

Fig. 3 Example of a file in the XES format

```
<log xes.version="1.0" xes.features="arbitrary-depth" xmlns="http://.../xes">
  <extension name="Concept" prefix="concept" uri="http://.../xes/concept.xesext"/>
  <extension name="Time" prefix="time" uri="http://.../xes/time.xesext"/>
  <global scope="trace">
    <string key="concept:name" value="" />
  </global>
  <global scope="event">
    <string key="concept:name" value="" />
    <date key="time:timestamp" value="1970-01-01T00:00:00.000+00:00" />
    <string key="resource" value="" />
  </global>
  <classifier name="Activity" keys="concept:name" />
  <float key="log attribute" value="2335.23" />
  <trace>
    <string key="concept:name" value="1" />
    <event>
      <string key="concept:name" value="Check stock availability" />
      <date key="time:timestamp" value="2012-07-30T11:14:00:000+01:00" />
      <string key="resource" value="SYS1" />
    </event>
    <event>
      <string key="concept:name" value="Retrieve product from warehouse" />
      <date key="time:timestamp" value="2012-07-30T14:20:00:000+01:00" />
      <string key="resource" value="Rick" />
    </event>
  </trace>
</log>
```

product from warehouse” on the abstraction level of a process model maps to a series of events like “work item #1,211 assigned,” “work item #1,211 started,” “purchase order form opened,” “product retrieved,” and “work item #1,211 completed.” Often, fine-granular events may show up repeatedly in the logs, while on an abstract level, only a single task is executed. Therefore, it is difficult to define a precise mapping between the two levels of abstraction. The problem of abstracting from low-level event data to activities in the business process model is proposed in Baier et al. (2014) by annotating, matching, and clustering. Other related techniques aim at discovering repetitive patterns of low-level events such that every occurrence of a discovered pattern can be abstracted as one occurrence of a higher-level event (Mannhardt and Tax 2017).

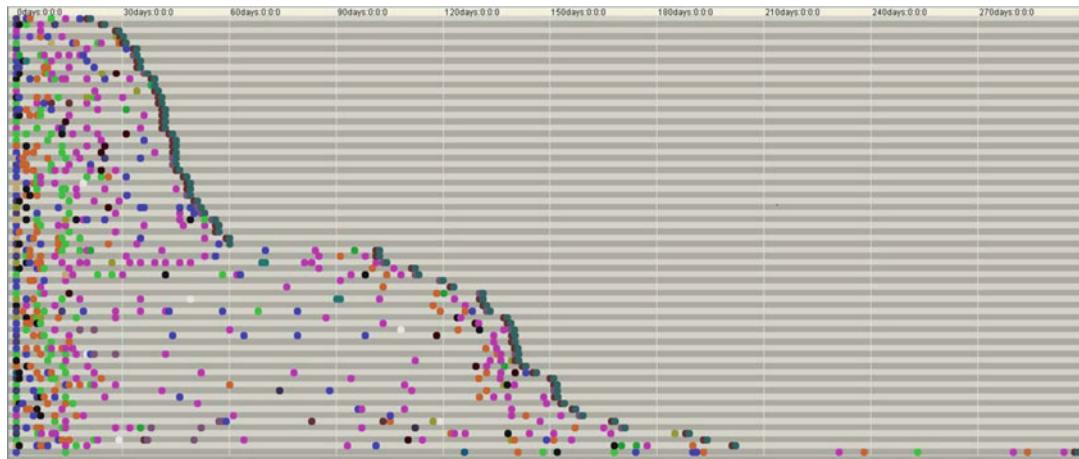
Once we have overcome the above data quality challenges, we are able to produce an event log consisting of *traces*, such that each trace contains the perfectly ordered sequence of events observed for a given case, each event refers to a task, and every task in the process is included in the event log. An event log with these prop-

erties can be rewritten in a simplified representation, called a *workflow log*. A workflow log is a set of distinct traces, such that each trace consists of a sequence of symbols and each symbol represents an execution of a task. For example, given a set of tasks represented by symbols $\{a, b, c, d\}$, a possible workflow log is $\{\langle a, b, c, d \rangle, \langle a, c, b, d \rangle, \langle a, c, d \rangle\}$.

Event Log Visualizations

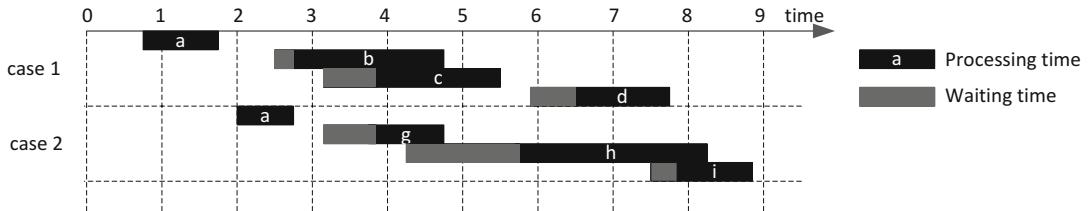
Visualization techniques for business process event logs can be broadly classified into two categories: *time-based* and *graph-based techniques*. Time-based techniques rely on charts where 1 on the axis (typically the horizontal axis) is used to denote the time. The vertical axis is then divided into stripes, each of which is used to represent the events observed in one trace of the log.

Graph-based techniques on the other hand show relations between tasks, resources, or possibly other objects involved in the process. Each node in the graph represents a task, resource,



B

Business Process Event Logs and Visualization, Fig. 4 Dotted chart of an event log



Business Process Event Logs and Visualization, Fig. 5 Example of timeline chart

or other object, while the edges represent control-flow relations (e.g., precedence).

Below are two representative time-based techniques, namely, dotted and timeline charts, and two graph-based techniques: dependency graphs and handoff graphs.

Dotted Charts

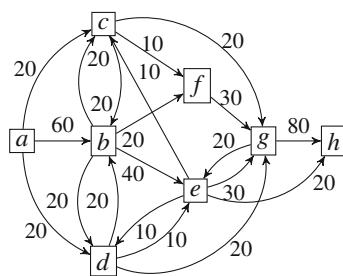
In a dotted chart (Song and van der Aalst 2007), each event is plotted on a two-dimensional canvas with the first axis representing its occurrence in time and the second axis as its association with a classifier like a case ID. There are different options to organize the first axis. Time can be represented either *relative*, such that the first event is counted as zero or *absolute* such that later cases with a later start event are further right in comparison to cases that started earlier. The second axis can be sorted according to different criteria. For instance, cases can be shown according to their historical order or their overall cycle time.

Figure 4 shows the dotted chart of the event log of a healthcare process. The events are plotted according to their relative time and sorted according to their overall cycle time. It can be seen that there is a considerable variation in terms of cycle time. Furthermore, the chart suggests that there might be three distinct classes of cases: those that take no more than 60 days, those taking 60 to 210 days, and a small class of cases taking longer than 210 days.

Timeline Charts

If the event log contains timestamps capturing both the start and the end of each task, we can plot tasks not as dots but as bars as shown in Fig. 5. This type of a visualization is called a *timeline chart*. A timeline chart shows the waiting time (from activation until starting) and the processing time (from starting until completion) for each task.

10 × a,b,c,g,e,h
 10 × a,b,c,f,g,h
 10 × a,b,d,g,e,h
 10 × a,b,d,e,g,h
 10 × a,b,e,c,g,h
 10 × a,b,e,d,g,h
 10 × a,c,b,e,g,h
 10 × a,c,b,f,g,h
 10 × a,d,b,e,g,h
 10 × a,d,b,f,g,h



Business Process Event Logs and Visualization, Fig. 6
Event log and corresponding dependency graph

Dependency Graphs

A *dependency graph* (also known as a *directly follows graph*) is a graph where each node represents one event class (i.e., a task) and each arc represents a “directly follows” relation. An arc exists between two event classes A and B if there is at least one trace in which B comes immediately after A. The arcs in a dependency graph may be annotated with an integer indicating the number of times that B directly follows A (hereby called the *absolute frequency*). In some process mining tools, it is also possible to annotate each arc with a temporal delay, e.g., the average time between an occurrence of task A and an occurrence of task B, among all occurrences of B that immediately follow an occurrence of A. This temporal delay gives an indication of the waiting time between a given pair of tasks in the process.

For example, Fig. 6 shows an event log (left) and a corresponding dependency graph (right). The event log consists of 10 distinct traces, each of which occurs 10 times in the log (hence 100 traces in total). We observe that the arc from task *a* to task *c* has an absolute frequency of 20. This is because the event log has two distinct traces where *c* occurs immediately before *a* and each of these distinct traces occurs 10 times.

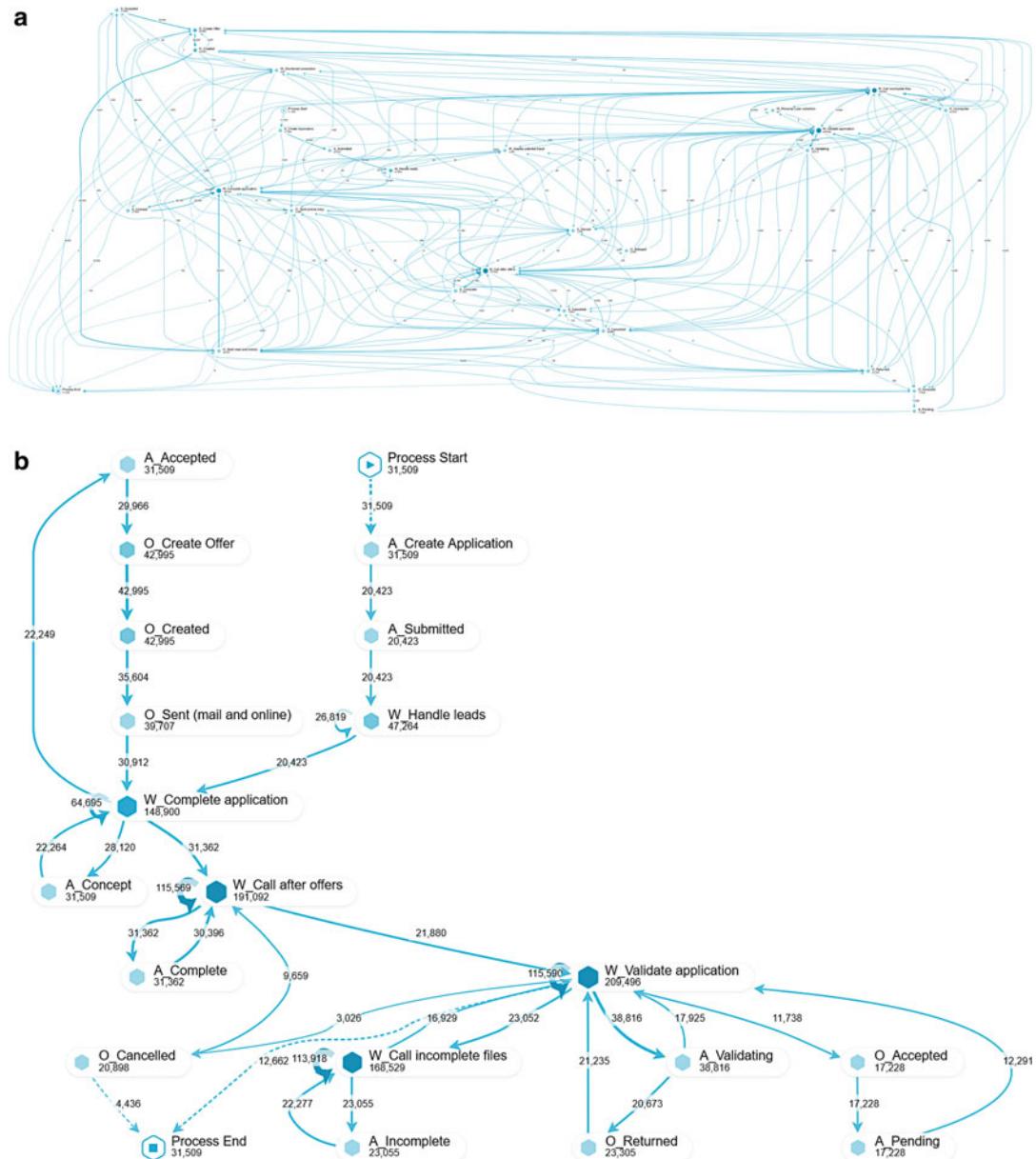
Owing to their simplicity, dependency graphs are supported by most commercial process mining tools. They are also supported by the *Fuzzy Miner* plugin of ProM (<http://www.promtools.org>) and by Apromore (<http://apromore.org>), two open-source process mining tool sets. All these tools provide visual cues to enhance the read-

ability of dependency graphs. For example, the color and the thickness of the arcs may be used to encode how frequently a task in a pair directly follows the other (e.g., a thick and dark-blue arc may represent a frequent directly follows relation relative to a thin, light-blue arc). These tools also offer visual zoom-in operations that allow users to explore specific parts of a dependency graph in detail.

Besides their simplicity, one of the most appealing features of dependency graphs is that they are amenable to *abstraction*. In this context, abstraction refers to removing a subset of the nodes or arcs in a dependency graph in order to obtain a smaller dependency graph of a given event log. For example, process mining tools allow us to remove the most infrequent nodes or arcs from a dependency graph in order to obtain a simpler map that is easier to understand. Abstraction is an indispensable feature when we want to explore a real-life event log, as illustrated below.

To illustrate how abstraction works, we consider the event log of the Business Process Intelligence Challenge 2017 (Available at: <https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b>). This is an event log of a loan origination process at a Dutch financial institution. The event log covers the application-to-approval and the offer-to-acceptance subprocesses of a loan origination process, meaning that it starts when a loan application is submitted by a customer and it ends when the customer accepts (or refuses) the corresponding loan offer. The full dependency graph produced by the Celonis web-based tool on this event log is shown in Fig. 7a. This map is too detailed to be understandable by a user. Even if we visually zoom in to inspect specific parts of this map, it will be very difficult to understand what is going on in this process. For this reason, process mining tools can abstract away the map by retaining only the most frequent arcs. Figure 7b shows the filtered map produced by Celonis when setting the task abstraction slider to 98% and the arc abstraction slider to 90%.

While abstraction is a useful mechanism when visualizing large event logs, it is not sufficient to handle the full complexity of real-life event



Business Process Event Logs and Visualization, Fig. 7 Example of a full dependency graph and an abstracted version thereof. (a) Full dependency graph. (b) Filtered dependency graph

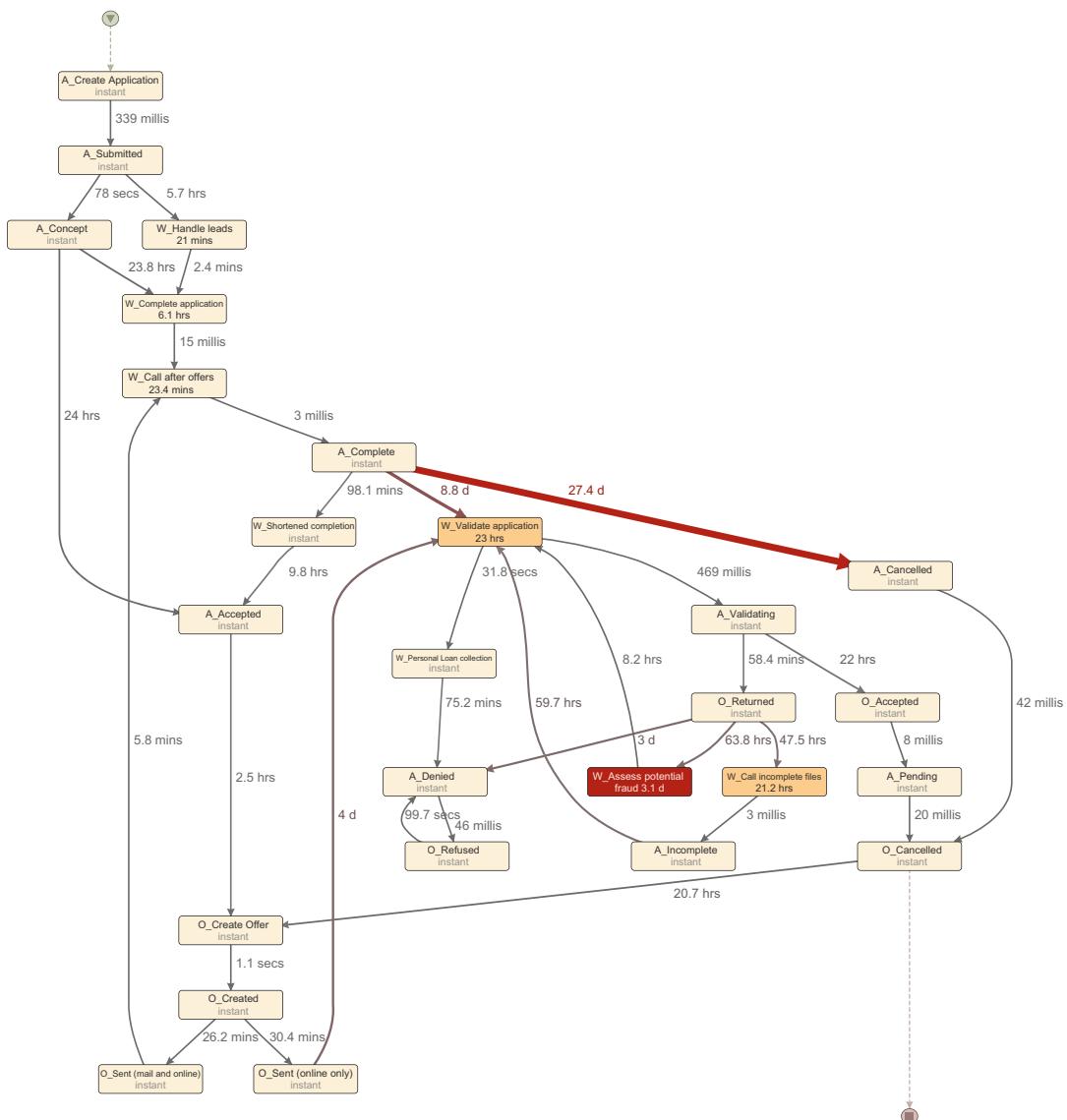
logs. Accordingly, process mining tools also offer a second type of simplification operation called *event log filtering*. Filtering an event log means removing a subset of the traces, events, or pairs of events in order to obtain a simpler log. Note that filtering and abstraction have different inputs and outputs. Whereas filtering transforms an event log

into a smaller log, abstraction only affects the dependency graph, not the log.

Dependency graphs can also be used to visualize temporal delays (e.g., waiting times or processing times) via color-coding or thickness-coding. Concretely, arcs in the dependency graph may be color-coded by the corresponding tem-

poral delay between the source event and the target event of the arc. This visualization allows us to identify the longest waiting times in the process – also known as *bottlenecks*. Meanwhile, the nodes in a dependency graph can be color-coded with the processing times of each task. The latter requires that each task captured in the log has both a start timestamp and an end timestamp so that the processing time can be computed.

Figure 8 shows the *performance view* produced by Disco when applied to the BPI Challenge 2017 log. The most infrequent arcs have been removed to avoid clutter. The task and the arc with the highest times are indicated in red. If we visually zoom in, in the tool, we are able to see that the task with the slowest processing time is “W_Assess Potential Fraud” (3.1 days) and the highest inter-event time is the one between “A_Complete” and “A_Cancelled” (27.4 days).



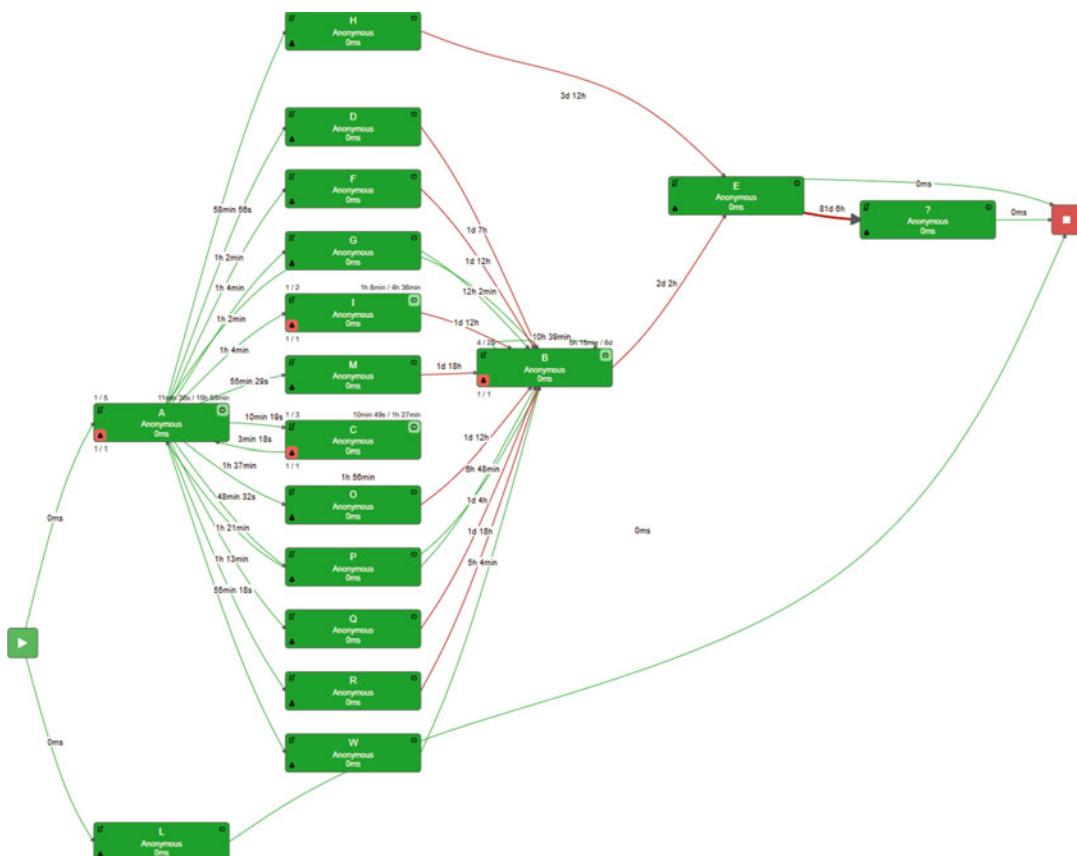
Business Process Event Logs and Visualization, Fig. 8 Performance view of the BPI Challenge 2017 event log in Disco

Handoff Graphs

Dependency graphs can also be used to analyze delays resulting from handoffs between process participants (van der Aalst and Song 2004; Pentland et al. 2017). When discussing the structure of event logs, we pointed out that an event record in an event log should have at least a case identifier, a timestamp, and an event class (i.e., a reference to a task in the process), but there can be other event attributes, too, like, for example, the *resource* (process participant) who performed the task in question. The event class is the attribute that is displayed in each node of a dependency graph. When importing an event log into a tool that supports dependency graphs, the tool gives an option to specify which event attribute should be used as the event class. We can set the tool so as to use the *resource* attribute as the event class. If we do so, the resulting dependency

graph will contain one node per resource (process participant), and there will be one arc between nodes A and B if resource B has performed a task immediately after resource A in at least one trace in the log. This map captures all the handoffs that have occurred in the process. We can then apply filters and abstraction operations to this map of handoffs. The handoff map can also be color-coded, with the colors corresponding either to the frequency of each handoff or the waiting time between handoffs.

Figure 9 shows a fragment of the *average duration view* produced by the myInvenio process mining tool when applied to an event log of a process for treating sepsis cases at a Dutch hospital (Log available at: <https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>). The handoff map shows that when a case starts (rightmost node), it is first handled



Business Process Event Logs and Visualization, Fig. 9 Handoff map

by resource A. After a few minutes or about an hour, resource A hands off to resource H or to resources D, F, ..., R. Resource H hands off to E, while , F, ..., R hand over to B who later hands off to E. Handoffs that take longer than one day are shown in red.

Pentland et al. 2017 propose an extension of dependency graphs (namely, *narrative networks*) for visualizing handoffs and identifying routines. The main idea is to show not only handoffs between process participants but also handoffs between systems (applications), for example, situations where the a process participant needs to use multiple systems to perform a task.

Other Log Visualization Techniques

Besides dotted and timeline charts, other examples of time-based log visualization techniques include *time series charts* and *cumulative flow diagrams*, which stem from the Kanban process management tradition (<https://kanbantool.com/kanban-library/introduction>). A time series chart is a line chart that displays the evolution of a quantitative variable over time. In the context of event logs, this variable is typically the number of ongoing cases – the so-called work-in-process (WIP). A point in the line represents the WIP at a time point. A cumulative flow diagram (CFD), on the other hand, shows the number of cases that have reached a given stage in the process (including cases that have reached the “completed” stage). In order to draw a CFD, we need to be able to decompose the tasks of the process into stages (Nguyen et al. 2016). Note that time series charts and cumulative flow charts are aggregate log visualization techniques: they do not display each event in the process, but rather they display how a quantity (e.g., WIP) evolves over time. In contrast, dotted and timeline charts display each event individually. Another aggregate log visualization technique, namely, rhythm-eyes views, uses a circular representation of time (Gulden 2016). Instead of time flowing linearly from left to right, it flows clockwise along a ring. Each segment of the ring denotes the mean cycle time of a given activity of the process.

As discussed above, dependency graphs can be simplified by means of abstraction operations,

which retain only the most frequent nodes and edges. Two alternative techniques for simplifying process maps are *node clustering* and *graph reduction*. In node clustering, a cluster of nodes is replaced by a single one. This approach is implemented in the Fuzzy Miner (Günther and van der Aalst 2007). In graph reduction techniques, the user designates a set of important nodes, and other nodes are then abstracted in such a way that the control-flow relations between the designated nodes remains visible (Daenen 2017). The groups of abstracted nodes are then replaced by special (thick) edges.

Cross-References

- ▶ [Business Process Querying](#)
- ▶ [Event Log Cleaning for Business Process Analytics](#)
- ▶ [Trace Clustering](#)

References

- Acampora G, Vitiello A, Stefano BND, van der Aalst WMP, Günther CW, Verbeek E (2017) IEEE 1849: the XES standard: the second IEEE standard sponsored by IEEE computational intelligence society [society briefs]. IEEE Comp Int Mag 12(2):4–8
- Baier T, Mendling J, Weske M (2014) Bridging abstraction layers in process mining. Inf Syst 46:123–139
- Bayomie D, Helal IMA, Awad A, Ezat E, ElBastawissi A (2015) Deducing case ids for unlabeled event logs. In: International conference on business process management. Springer, pp 242–254
- Daenen K (2017) Path-colored flow diagrams: increasing business process insights by visualizing event logs. In: Proceedings of the 15th international conference on business process management (BPM). LNCS, vol 10445. Springer, pp 94–109
- Gulden J (2016) Visually comparing process dynamics with rhythm-eye views. In: Proceedings of the business process management workshops. LNBIP, vol 281. Springer, pp 474–485
- Günther CW, van der Aalst WMP (2007) Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: 5th international conference on business process management (BPM). LNCS, vol 4714. Springer, pp 328–343
- Mannhardt F, Tax N (2017) Unsupervised event abstraction using pattern abstraction and local process models. CoRR, abs/1704.03520

- Nguyen H, Dumas M, ter Hofstede AHM, Rosa ML, Maggi FM (2016) Business process performance mining with staged process flows. In: Proceedings of the 28th international conference on advanced information systems engineering (CAiSE). LNCS, vol 9694. Springer, pp 167–185
- Pentland BT, Recker J, Wyner G (2017) Rediscovering handoffs. *Acad Manag Discov* 3(3):284–301
- Pourmirza S, Dijkman R, Grefen P (2017) Correlation miner: mining business process models and event correlations without case identifiers. *Int J Coop Inf Syst* 26(2):1742002
- Solti AR, Mans RS, van der Aalst WMP, Weske M (2013) Improving documentation by repairing event logs. In: IFIP working conference on the practice of enterprise modeling. Springer, pp 129–144
- Song M, van der Aalst WMP (2007) Supporting process mining by showing events at a glance. In: Proceedings of the 17th annual workshop on information technologies and systems (WITS), pp 139–145
- Suriadi S, Andrews R, ter Hofstede AHM, Wynn MT (2017) Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. *Inf Syst* 64:132–150
- van der Aalst WMP, Song M (2004) Mining social networks: uncovering interaction patterns in business processes. In: Proceedings of the 2nd international conference on business process management (BPM). Lecture notes in computer science, vol 3080. Springer, pp 244–260

Business Process Execution Log

- ▶ [Business Process Event Logs and Visualization](#)

Business Process Execution Trail

- ▶ [Business Process Event Logs and Visualization](#)

Business Process Mining

- ▶ [Business Process Analytics](#)

Business Process Model Matching

B

Henrik Leopold

Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Synonyms

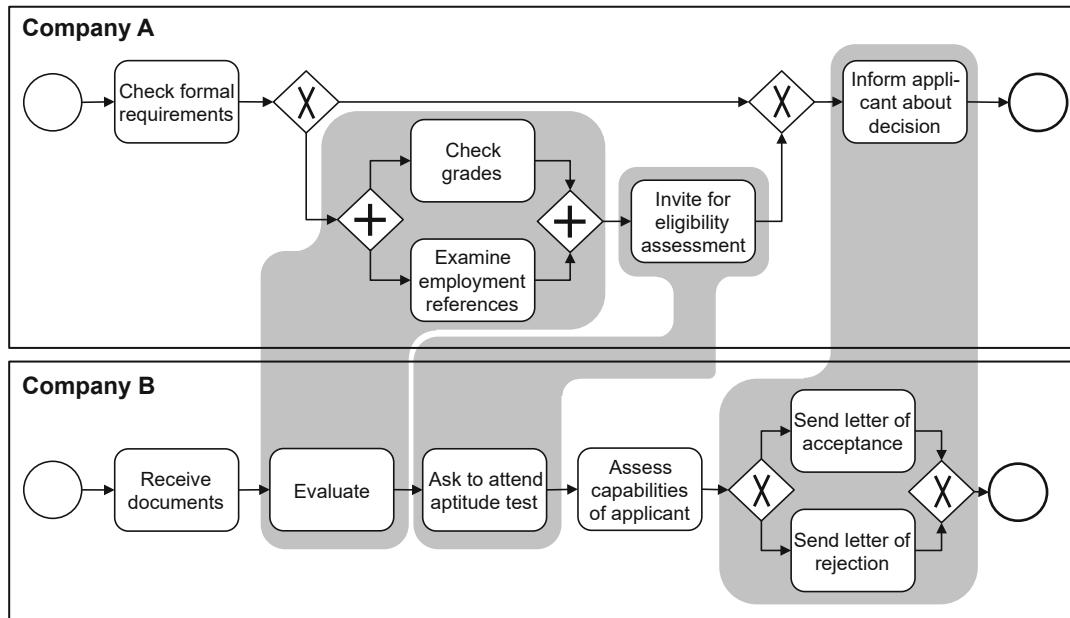
[Alignment creation](#); [Process matching](#)

Definitions

The comparison and integration of different representations of reality, which is often referred to as *matching*, is a long-standing problem in computer science. Matching techniques have been developed for various representations, including database schemas (Rahm and Bernstein 2001; Gal 2011; Sagi and Gal 2013), ontologies (Noy 2004; Shvaiko and Euzenat 2013; Euzenat et al. 2007), and, more recently, also business process models (Dijkman et al. 2009; Weidlich et al. 2010; Leopold et al. 2012). In the context of process models, matching techniques aim to automatically identify activity correspondences that represent similar behavior in both models.

Overview

To get an understanding of the challenges that are associated with business process model matching, consider Fig. 1. It shows two recruiting processes from two different companies. The gray shades denote the activity correspondences for the two models. For example, the activity *Evaluate* from company B corresponds to the activities *Check grades* and *Examine employment references* from company A. The correspondences show that the two models differ with respect to the terms they use (e.g., *eligibility assessment* versus *aptitude test*) as well as their level of detail (e.g., *Evaluate* is described in more detail in the model from company A). Given such differences, the proper recognition of the correspondences



Business Process Model Matching, Fig. 1 Two process models and possible correspondences

between two process models can become a complex and challenging task (Mendling et al. 2015). The main difference to other matching problems, such as schema and ontology matching, is the temporal dimension of process models. Two activities with the exact same label may refer to different streams of action in reality if they occur at different moments of time in the process.

Key Research Findings

The research in the area of business process model matching has focused on two main aspects: defining automatic techniques for process model matching and developing appropriate mechanisms to evaluate these techniques. This section discusses the key findings from both streams.

Process Model Matching Techniques

Current process model matching techniques exploit a variety of techniques to automatically identify activity correspondences. Most of them, however, leverage either the activity labels or the model structure of process models.

Leveraging Activity Labels

Activity labels capture the main semantics of process models. Hence, they are also the primary source of process model matching techniques for identifying activity correspondences. However, the way in which the similarity between two activity labels is quantified differs considerably. There are two types of measures that can be employed: syntactic measures and semantic measures.

Syntactic measures relate to simple string comparisons and do not take the meaning or context of words into account. The most prominently employed syntactic measure is the *Levenshtein distance*. Given two labels l_1 and l_2 , the Levenshtein distance counts the number of edit operations (i.e., insertions, deletions, and substitutions) that are required to transform l_1 into l_2 . Although the Levenshtein distance is rather simplistic, it is used by a variety of matching techniques (cf. Dijkman et al. 2009; Liu et al. 2014; Belhoul et al. 2012). Many matching techniques, however, also rely on plain word comparisons (cf. Fengel 2014; Antunes et al. 2015; Cayoglu et al. 2013). Very common measures include the *Jaccard* and the *Dice*

coefficient, which both compute the similarity between two activity labels based on the number of shared words. An alternative approach based on word comparisons is the *cosine similarity* (cf. Weidlich et al. 2010; Makni et al. 2015). To compute the cosine similarity, activity labels are transformed into vectors, typically by weighing words with their frequency of occurrence. The cosine similarity is then derived from the cosine of the angle between the two activity vectors.

Semantic measures take the meaning of words into account. A very common strategy to do so is the identification of synonyms using the lexical database WordNet (Miller 1995). Typically, matching techniques check for synonyms as part of a prepossessing step and then apply other, often also syntactic, similarity measures (cf. Antunes et al. 2015). The most prominent semantic measure is the Lin similarity (Lin 1998). The Lin similarity is a method to compute the semantic relatedness of words based on their information content according to the WordNet taxonomy. To use Lin for measuring the similarity between two activities (which mostly contain more than a single word), matching techniques typically combine the Lin similarity with the bag-of-words model (see e.g., Klinkmüller et al. 2013; Leopold et al. 2012). The bag-of-words model transforms an activity into a multiset of words that ignores grammar and word order. The Lin similarity can then be obtained by computing the average of the word pairs from the two bags with the highest Lin score. Other measures based on the WordNet dictionary include the ones proposed by Wu and Palmer (1994) and Lesk (1986). The former computes the similarity between two words by considering the path length between the words in the WordNet taxonomy. The latter compares the WordNet dictionary definitions of the two words. Some approaches also directly check for hypernym relationships (i.e., more common words). For instance, Hake et al. (described in Antunes et al. 2015) consider *car* and *vehicle* as identical words because *vehicle* is a hypernym of *car*.

Leveraging Model Structure

Despite of the predominant role of activity labels for identifying correspondences, many pro-

cess model matching techniques also exploit the model structure. Typically, structural features of the model are used to guide the matching process based on the computed activities' similarities or to rule out unlikely constellations.

One of the most prominent strategies to take the model structure into account is the *graph edit distance* (cf. Dijkman et al. 2009; Gater et al. 2010). It applies the notion of string edit distance as used by the Levenshtein distance to graphs. In the context of process model matching, the graph edit distance is used to obtain a set of correspondences that is as similar as possible from a structural perspective. An alternative way of taking the structure in account is to leverage the notion of single-entry-single-exit fragments. As described by Vanhatalo et al. (2009), a process model can be decomposed into a hierarchy of fragments with a single entry and a single exit node. The resulting fragment hierarchy is useful for both identifying candidates for one-to-many relationships as well as identifying correspondences that have similar depth in the model. Notable examples exploiting the fragment hierarchy include the ICoP Framework from Weidlich et al. (2010) and the matching techniques from Gerth et al. (2010) and Branco et al. (2012).

Some approaches also use the structure of a process model to derive constraints. For instance, Leopold et al. (2012) derive behavioral constraints from the model structure that must not be violated in the final set of correspondences. That means if two activities a_1 and a_2 from model A are in a strict order relation and two activities b_1 and b_2 from model B are in a strict order relation as well, the final set of correspondences cannot include a correspondence between a_1 and b_2 and between a_2 and b_1 . A similar notion is used in the matching approach from Meilicke et al. (2017).

Alternative Techniques

Most existing process model matching techniques combine the algorithms discussed above in a specific way. Some techniques, such as the ICoP framework (Weidlich et al. 2010), also allow the user to configure the matching technique by selecting the desired

algorithms. However, there are also techniques that consider alternative strategies to identify activity correspondences. More specifically, there are techniques building on user feedback, prediction, and matching technique ensembles.

Klinkmüller et al. (2014) proposed a semiautomated technique that incorporates *user feedback* to improve the matching performance. The main idea of their approach is to present the automatically computed correspondences to the user, let the user correct incorrect correspondences, and then, based on the feedback, adapt the matching algorithm. Weidlich et al. (2013) proposed a technique that aims at *predicting* the performance of a matching technique based on the similarity values it produces for a given matching problem. In this way, the approach can help to select the most suitable matching technique for the problem at hand. Meilicke et al. (2017) proposed an *ensemble-based* technique that runs a set of available matching systems and then selects the best correspondences based on a voting scheme and a number of behavioral constraints. This strategy has the advantage of combining the strengths of multiple matching techniques.

While all these techniques are able to improve the matching results in certain settings, they also come with their own limitations. Incorporating user feedback as proposed by Klinkmüller et al. (2014) means that the results can no longer be obtained in a fully automated fashion. The process model prediction technique from Weidlich et al. (2013) may simply conclude that available techniques are not suitable for computing correspondences. In the same way, the ensemble-matching technique from Meilicke et al. (2017) relies on the availability of matching techniques that can actually identify the complex relationships that may exist between two models.

Approaches for Evaluating Process Model Matching Techniques

The evaluation of process model matching techniques is a complex task. To illustrate this, reconsider the correspondences from Fig. 1. While

one can argue that the activity *Invite for eligibility assessment* from Company A corresponds to the activity *Ask to attend aptitude test* because both activities have a similar goal, one can also bring up arguments against it. We could, for instance, argue that the *eligibility assessment* in the process of Company A is optional, while the *aptitude test* in the process of Company B is mandatory. What is more, an *aptitude test* appears to be way more specific than an *eligibility assessment*. This example illustrates that correspondences might be actually disputable and that clearly defining which correspondences are correct can represent a complex and challenging task. This is, for instance, illustrated by the gold standards of the Process Model Matching Contests 2013 and 2015. The organizers of the contests found that there was not a single model pair for which two independent annotators fully agreed on all correspondences (Cayoglu et al. 2013; Antunes et al. 2015).

The standard procedure for evaluating process model matching techniques builds on a *binary gold standard*. Such a gold standard is created by humans and clearly defines which correspondences are correct. By comparing the correspondences produced by the matching technique and the correspondences from the gold standard, it is then possible to compute the well-established metrics precision, recall, and F measure. At this point, the evaluation based on a binary gold standard is widely established and was also used by two Process Model Matching Contests. The advantage of using a binary gold standard is that the notion of precision, recall, and F measure is well-known and easy to understand. However, it also raises the question why the performance of process model matching techniques is determined by referring to a single correct solution when not even human annotators can agree what this correct solution is. A binary gold standard implies that any correspondence that is not part of the gold standard is incorrect and, thus, negatively affects the above mentioned metrics.

Recognizing the limitations of a binary gold standard, Kuss et al. (2016) recently introduced

the notion of a *non-binary gold standard*. The overall rationale of a non-binary gold standard is to take the individual opinion of several people independently into account. Therefore, several people are asked to create a binary gold standard based on their own opinion. By considering each individual opinion as a vote for a given correspondence, it is then possible to compute a support value for each correspondence between 0.0 and 1.0. Based on these support values, Kuss et al. (2016) proposed non-binary notions of the metrics precision, recall, and F measure that take the uncertainty of correspondences into account. While the resulting metrics are not as intuitive to interpret as their binary counterparts, they can be considered as a fairer assessment of quality of a matching technique. Following that argument, the results from non-binary evaluation were recently presented next to the results of the binary evaluation in the Process Model Matching Track of the Ontology Alignment Evaluation Initiative 2016 (Achichi et al. 2016).

Examples of Application

The activity correspondences produced by process model matching techniques often serve as input for advanced process model analysis techniques. Examples include the analysis of model differences (Küster et al. 2006), the harmonization of process model variants (La Rosa et al. 2013), the process model search (Jin et al. 2013), and the detection of process model clones (Uba et al. 2011). Many of these techniques require activity correspondences as input and, therefore, implicitly build on process model matching techniques. However, the application of process model matching techniques also represents an analysis by itself. The identified correspondences inform the user about similarities and dissimilarities between two models (or sets of models). From a practical perspective, process model matching techniques can, among others, provide valuable insights for identifying overlap or synergies in the context of

company mergers or for analyzing the evolution of models.

B

Future Directions for Research

As indicated by the results of the two Process Model Matching Contests, the main challenge in the area of process model matching remains the rather moderate and unstable performance of the matching techniques. The best F measures in the Process Model Contest 2015 ranged between 0.45 and 0.67. A recent meta-analysis by Jabeen et al. (2017) suggests that this can be mainly explained by the predominant choice for syntactic and simplistic semantic activity label similarity measures. These measures are simply not capable of identifying complex semantic relationships that exist between process models. These observations lead to three main directions for future research:

1. *Identify better strategies to measure activity label similarity:* There is a notable need for more intelligent strategies to measure the similarity between activity labels. Syntactic similarity measures are useful for recognizing correspondences with identical or very similar labels. The sole application of these measures, however, has been found to simply result in a high number of false positives (Jabeen et al. 2017). Also the current use of semantic technology is insufficient. Comparing activity labels by computing the semantic similarity between individual word pairs does not account for the cohesion and the complex relationships that exist between the words of activity labels. Here, a first step would be the proper consideration of compound nouns such as *eligibility assessment*. A more advanced step would be to also consider the relationship between nouns and verbs (Kim and Baldwin 2006). The fact that two activities use the same verb does not necessarily imply that they are related. Possible directions to account for these complex relationships are technologies such as distributional semantics (Bruni et al.

- 2014). They have been found to considerably improve the results in other matching contexts (Leopold et al. 2015).
2. *Incorporate machine learning:* Existing process model matching techniques hardly consider the use of machine learning. However, in the domain of ontology and schema matching, this seems to be a promising direction (cf. Doan et al. 2004; Spohr et al. 2011; Berlin and Motro 2002). Looking into the shortcomings of existing matching techniques, it seems that machine learning could help in two specific ways. First, they could be used to improve the detection of relationships between domain-specific words. This could be accomplished by training matching techniques on domain-specific dictionaries or other textual resources. Second, they could be used to better filter out unlikely correspondences. While first approaches have exploited simple behavioral rules to avoid certain correspondence constellations (cf. Leopold et al. 2012), the actual patterns might be more complex and context-dependent. Machine learning techniques might be able to detect those.
 3. *Increase the focus on the use case:* Currently, process model matching techniques are designed as general-purpose tools and do not take into account how the generated correspondences will be used later on. Depending on the actual use case, however, both the level of acceptable human interaction as well as the way how the technique needs to be evaluated might change. Consider, for example, the use case of identifying the operational overlap between two large process model repositories of two different organizations. While this use case certainly requires a fully automated solution, it probably does not need to be perfectly accurate in terms of precision. The manual work that is saved by pointing at potentially similar processes makes the matching technique already useful in a practical setting. This example shows that it might be worth tailoring matching techniques to specific use cases instead of designing general-purpose tools.

Cross-References

- [Business Process Querying](#)
- [Graph Pattern Matching](#)
- [Holistic Schema Matching](#)
- [Uncertain Schema Matching](#)

References

- Achichi M, Cheatham M, Dragisic Z, Euzenat J, Faria D, Ferrara A, Flouris G, Fundulaki I, Harrow I, Ivanova V et al (2016) Results of the ontology alignment evaluation initiative 2016. In: CEUR workshop proceedings, RWTH, vol 1766, pp 73–129
- Antunes G, Bakshandeh M, Borbinha J, Cardoso J, Dadashnia S, Francescomarino CD, Dragoni M, Fettke P, Gal A, Ghidini C, Hake P, Khiat A, Klinkmüller C, Kuss E, Leopold H, Loos P, Meilicke C, Niesen T, Pesquita C, Péus T, Schoknecht A, Sheetrit E, Sonntag A, Stuckenschmidt H, Thaler T, Weber I, Weidlich M (2015) The process model matching contest 2015. In: 6th international workshop on enterprise modelling and information systems architectures
- Belhou Y, Haddad M, Duchêne E, Khedouci H (2012) String comparators based algorithms for process model matchmaking. In: 2012 IEEE ninth international conference on services computing (SCC). IEEE, pp 649–656
- Berlin J, Motro A (2002) Database schema matching using machine learning with feature selection. In: International conference on advanced information systems engineering. Springer, pp 452–466
- Branco MC, Troya J, Czarnecki K, Küster J, Völzer H (2012) Matching business process workflows across abstraction levels. In: Proceedings of the 15th international conference on model driven engineering languages and systems, MODELS’12. Springer, Berlin/Heidelberg, pp 626–641
- Bruni E, Tran NK, Baroni M (2014) Multimodal distributional semantics. J Artif Intell Res (JAIR) 49(2014):1–47
- Cayoglu U, Dijkman R, Dumas M, Fettke P, García-Bañuelos L, Hake P, Klinkmüller C, Leopold H, Ludwig A, Loos P et al (2013) The process model matching contest 2013. In: 4th international workshop on process model collections: management and reuse (PMC-MR’13)
- Dijkman RM, Dumas M, García-Bañuelos L (2009) Graph matching algorithms for business process model similarity search. In: BPM, vol 5701. Springer, pp 48–63
- Doan A, Madhavan J, Domingos P, Halevy A (2004) Ontology matching: a machine learning approach. In: Staab S, Studer R (eds) Handbook on ontologies. International Handbooks on Information Systems. Springer, Berlin/Heidelberg, pp 385–403

- Euzenat J, Shvaiko P et al (2007) Ontology matching, vol 18. Springer, Berlin/Heidelberg
- Fengel J (2014) Semantic technologies for aligning heterogeneous business process models. *Bus Process Manag* J 20(4):549–570
- Gal A (2011) Uncertain schema matching. *Synth Lect Data Manag* 3(1):1–97
- Gater A, Grigori D, Bouzeghoub M (2010) Complex mapping discovery for semantic process model alignment. In: Proceedings of the 12th international conference on information integration and web-based applications & services. ACM, pp 317–324
- Gerth C, Luckey M, Küster J, Engels G (2010) Detection of semantically equivalent fragments for business process model change management. In: 2010 IEEE international conference on services computing (SCC). IEEE, pp 57–64
- Jabeen F, Leopold H, Reijers HA (2017) How to make process model matching work better? an analysis of current similarity measures. In: Business information systems (BIS). Springer
- Jin T, Wang J, La Rosa M, Ter Hofstede A, Wen L (2013) Efficient querying of large process model repositories. *Comput Ind* 64(1):41–49
- Kim SN, Baldwin T (2006) Interpreting semantic relations in noun compounds via verb semantics. In: Proceedings of the COLING/ACL on main conference poster sessions. Association for Computational Linguistics, pp 491–498
- Klinkmüller C, Weber I, Mendling J, Leopold H, Ludwig A (2013) Increasing recall of process model matching by improved activity label matching. In: Business process management. Springer, pp 211–218
- Klinkmüller C, Leopold H, Weber I, Mendling J, Ludwig A (2014) Listen to me: improving process model matching through user feedback. In: Business process management. Springer, pp 84–100
- Kuss E, Leopold H, Van der Aa H, Stuckenschmidt H, Reijers HA (2016) Probabilistic evaluation of process model matching techniques. In: Conceptual modeling: proceedings of 35th international conference, ER 2016, Gifu, 14–17 Nov 2016. Springer, pp 279–292
- Küster JM, Koehler J, Ryndina K (2006) Improving business process models with reference models in business-driven development. In: Business process management workshops. Springer, pp 35–44
- La Rosa M, Dumas M, Uba R, Dijkman R (2013) Business process model merging: an approach to business process consolidation. *ACM Trans Softw Eng Methodol (TOSEM)* 22(2):11
- Leopold H, Niepert M, Weidlich M, Mendling J, Dijkman R, Stuckenschmidt H (2012) Probabilistic optimization of semantic process model matching. In: Business process management. Springer, pp 319–334
- Leopold H, Meilicke C, Fellmann M, Pittke F, Stuckenschmidt H, Mendling J (2015) Towards the automated annotation of process models. In: International conference on advanced information systems engineering. Springer, pp 401–416
- Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: Proceedings of the 5th annual international conference on systems documentation. ACM, pp 24–26
- Lin D (1998) An information-theoretic definition of similarity. In: ICML, vol 98, pp 296–304
- Liu K, Yan Z, Wang Y, Wen L, Wang J (2014) Efficient syntactic process difference detection using flexible feature matching. In: Asia-Pacific conference on business process management. Springer, pp 103–116
- Makni L, Haddar NZ, Ben-Abdallah H (2015) Business process model matching: an approach based on semantics and structure. In: 2015 12th international joint conference on e-business and telecommunications (ICETE), vol 2. IEEE, pp 64–71
- Meilicke C, Leopold H, Kuss E, Stuckenschmidt H, Reijers HA (2017) Overcoming individual process model matcher weaknesses using ensemble matching. *Decis Support Syst* 100(Supplement C):15–26
- Mendling J, Leopold H, Pittke F (2015) 25 challenges of semantic process modeling. *Int J Inf Syst Softw Eng Big Co (IJISEBC)* 1(1):78–94
- Miller GA (1995) WordNet: a lexical database for English. *Commun ACM* 38(11):39–41
- Noy NF (2004) Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Rec* 33(4):65–70
- Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. *VLDB J* 10(4):334–350
- Sagi T, Gal A (2013) Schema matching prediction with applications to data source discovery and dynamic ensembling. *VLDB J Int J Very Large Data Bases* 22(5):689–710
- Shvaiko P, Euzenat J (2013) Ontology matching: state of the art and future challenges. *IEEE Trans Knowl Data Eng* 25(1):158–176
- Spohr D, Hollink L, Cimiano P (2011) A machine learning approach to multilingual and cross-lingual ontology matching. In: The semantic web-ISWC 2011, pp 665–680
- Uba R, Dumas M, García-Bañuelos L, La Rosa M (2011) Clone detection in repositories of business process models. In: Business process management. Springer, pp 248–264
- Vanhalatalo J, Völzer H, Koehler J (2009) The refined process structure tree. *Data Knowl Eng* 68(9):793–818
- Weidlich M, Dijkman R, Mendling J (2010) The ICO framework: identification of correspondences between process models. In: Advanced information systems engineering. Springer, pp 483–498
- Weidlich M, Sagi T, Leopold H, Gal A, Mendling J (2013) Predicting the quality of process model matching. In: Business process management. Springer, pp 203–210
- Wu Z, Palmer M (1994) Verbs semantics and lexical selection. In: Proceedings of the 32nd annual meeting on association for computational linguistics. Association for Computational Linguistics, pp 133–138

Business Process Monitoring

► Business Process Analytics

Business Process Performance Measurement

Adela del-Río-Ortega, Manuel Resinas, and Antonio Ruiz-Cortés

University of Seville, Sevilla, Spain

Definitions

Business process performance measurement, commonly shortened to process performance measurement (PPM), aims at checking the achievement of strategic and operational goals and supporting decision-making for the continuous optimization of business processes. It involves the definition, gathering, visualization, and analysis of a set of indicators that evaluate performance-relevant data of one or several business processes. These indicators, usually known as process performance indicators (PPIs), are quantifiable metrics that allow an evaluation of the efficiency and effectiveness of business processes and can be measured directly by data generated within the process flow. For instance, one could have a PPI for an order-to-cash process that specifies that its cycle time should be less than 20 working days in order to keep customer satisfaction.

Overview

Traditionally, process performance measurement (PPM) focused on the strategic and tactical level most frequently taking into account only financial measures. However, toward the early 1990s, a more balanced and integrated approach that involved the identification and use of performance indicators emerged due to the challenges posed by the rapidly changing technological and so-

cietal environment (Van Looy and Shafagatova 2016).

In this context, new definitions of performance measurement models arose. They intend to provide guidelines to evaluate the performance of an organization or a part of it, by considering different performance perspectives for which specific PPIs should be further defined. Depending on their focus, two categories can be distinguished. Organizational performance measurement models, like the balance scorecard (Kaplan and Norton 1996) or the European Foundation for Quality Management excellence model (EFQM 2010), provide a holistic view of an organization's performance and usually include financial and non-financial measures (related to the customer and other stakeholders, to processes and products, or to the strategy, among others). On the other hand, business process performance measurement models, such as statistical process control or the Devil's Quadrangle (Jansen-Vullers et al. 2008), have generally a less holistic focus, more specific to business processes, and include performance dimensions like time, cost, quality, or flexibility.

Defining PPIs

Regardless of the performance perspective to which a PPI applies, a collection of attributes have to be specified in its definition, namely, an *identifier* to uniquely designate the PPI, a descriptive *name* that helps to identify the PPI in an user-friendly manner, the *process* to which the PPI is related to or defined over, the strategic and/or operational *goals* with which the PPI is aligned, its *measure definition* that describes the way in which the indicator has to be calculated, a *target* value that must be reached to meet the pursued goals, and a *scope* that specifies the subset of process instances that must be considered to compute the PPI value and is frequently, though not necessarily, defined as a time frame. Furthermore, there are some other attributes that can be defined and provide additional information. Some examples are the *source of information* that refers to the data source used to gather the information to calculate the PPI, for instance, an event log or attributes related to some involved

human resources like *responsible*, *accountable*, or *informed*.

When defining PPIs with the just-introduced attributes, there is a set of requirements that need to be fulfilled. (1) A PPI must satisfy the SMART criteria, which stands for: specific, it has to be clear what the indicator exactly describes; measurable, it has to be possible to measure a current value and to compare it to the target one; achievable, its target must be challenging but achievable because it makes no sense to pursue a goal that will never be met; relevant, it must be aligned with a part of the organization's strategy, something that really affects its performance; and time-bounded, a PPI only has a meaning if it is known in the time period in which it is measured so that its evolution in time can be evaluated. (2) The PPIs themselves must be efficient, i.e., since the measurement itself requires human, financial, and physical resources, it must be worth the effort from a cost/benefit viewpoint. In addition, the amount of PPIs to define also must be efficient, following the less-is-more rule, where a balance between information need and information overload has to be met. (3) PPIs must be defined in an understandable way, according to the vocabulary used in the specific domain and in a language comprehensible by the different stakeholders involved in the PPI management. (4) They must keep traceability to the business process they are defined for, to avoid consistency issues when, for instance, the business process evolves but PPI definitions are not consequently updated and become obsolete. (5) They must be amenable to automation in order to avoid the need for a redefinition to a computable language, which could lead to the introduction of errors in the redefinition process.

Types of PPIs

PPIs can be broadly classified into two categories, namely, *lag* and *lead indicators*, also known as outcomes and performance drivers, respectively. The former establishes a goal that the organization is trying to achieve and is usually linked to a strategic goal. For instance, the previously introduced PPI example for the order-to-cash process: its cycle time should be less than 20 working

days in order to keep customer satisfaction. The problem of lag indicators is that they tell the organization whether the goal has been achieved, but they are not directly influenceable by the performers of the process. On the contrary, lead indicators have two main characteristics. First, they are predictive in the sense that if the lead indicators are achieved, then it is likely that the lag indicator is achieved as well. Second, they are influenceable by the performers of the process, meaning that they should refer to something that the performers of the process can actively do or not do (McChesney et al. 2012). For instance, if we think that one major issue that prevents fulfilling the aforementioned lag indicator is missing information from the customer when the order is created, double checking with the customer the order within the first 24 h could be a lead indicator for such lag indicator. Furthermore, it is important to distinguish PPIs from other similar concepts like activity metrics or risk indicators. Activity metrics measure business activity related to PPIs but do not have targets and goals associated with them because it does not make sense by definition, for instance, "top ten customers by revenue." They provide additional context that helps business people to make informed decisions. Finally, risk indicators measure the possibility of future adverse impact. They provide an early sign to identify events that may harm the continuity of existing processes (Van Looy and Shafagatova 2016).

Evaluating PPIs

Once a system of PPIs has been defined taking into consideration all the aforementioned information, data sources, methods, and instruments need to be established in order to gather the required data to calculate their values. Gathering methods can be classified into two big groups: evidence-based methods and subjective methods. The former includes the analysis of existing documentation, the observation of the process itself, and the use of data recorded by existing information systems that support the processes. The latter includes interviews, questionnaires, or workshops, where domain experts are queried to obtain the required information.

Evidence-based methods are less influenced by the individuals involved in the measurement than subjective ones; however they come with some weaknesses: people acting different because they are being observed, the existence of inaccurate or misleading information, or the unavailability of important data. Therefore, sometimes it can be a good decision to use a combination of them.

Analysis with PPIs

The last part of PPM is the visualization and analysis of performance data. This visualization and analysis serve two purposes. First, monitoring the process performance to keep it under control and to be able to quickly learn whether the performance is moving to the right direction so that appropriate actions are taken. Second, diagnosing the process performance in order to be able to spot points of potential improvement. Consequently, the techniques used to visualize and analyze the process performance vary depending on its purpose.

Dashboards are mainly used for the first purpose. According to Few (2006), dashboards are visual displays of the most important information needed to achieve one or more objectives that fit on a single computer screen so it can be monitored at a glance. Therefore, in PPM, dashboards are used for presenting the PPIs and their degree of fulfilment. In addition, they can depict other activity metrics or historic information that are necessary to give PPIs the context that is necessary to achieve the objectives for which the dashboard has been designed.

On the contrary, visual interfaces designed for data exploration and analysis are used for diagnosing the process performance. These interfaces offer all sorts of facilities to filter, group, or drill down data and to create different types of graphs that enable the comparison and analysis that are necessary to make sense of the performance data. Also, techniques such as statistical hypothesis testing or multivariate analysis and methods such as statistical process control can complement these visual interfaces to help identify causes of process performance variation.

Usually these two purposes go hand in hand. Ideally, visualization tools should have the ability

to go from the dashboard to an environment for exploration and analysis to learn insights from data and take action.

Regardless of the purpose, visualization designers should exploit well-known knowledge on visual perception to create effective visualizations that serve the aforementioned purposes. This knowledge include taking into account the limited capacity of people to process visual information: the use of color, form, position, and motion to encode data for rapid perception and the application of the Gestalt principles of visual perception that reveal the visual characteristics that incline us to group objects together (Few 2006). Furthermore, especially in the case of dashboards, simplicity is a key aspect to strive for since dashboards require to squeeze a great amount of information in a small amount of space while preserving clarity.

Key Research Findings

Performance Measurement Models

One specific stream of work that is particularly relevant to PPM is that related to performance measurement frameworks or models. Without a doubt, one of the most widely recognized performance measurement models is the balanced scorecard proposed by Kaplan and Norton (1996). They introduced a four-dimensional approach: (1) financial perspective, (2) customer perspective, (3) internal business process perspective, and (4) learning and growth perspective. It is, however, only one of several proposals in this area. For example, Keagan et al. (1989) proposed their performance measurement matrix which tries to integrate different business performance perspectives: financial and nonfinancial and internal and external. Brignall et al. (1991) developed the results and determinants framework, based on the idea of the two types of PPIs, lag, which relate to results (competitiveness, financial performance), and lead indicators, which relate to determinants (quality, flexibility, resource utilization, and innovation). The performance pyramid presented by Lynch and Cross (1991)

considers four levels, from top to down: (1) vision; (2) market and financial; (3) customer satisfaction, flexibility, and productivity; and (4) quality, delivery, cycle time, and waste. Adams and Neely (2002) used a performance prism with five facets to represent their proposal: stakeholder satisfaction, strategies, processes, capabilities, and stakeholder contribution. Another popular model is EFQM (2010), which distinguishes between enablers such as leadership, people, strategy, or processes, among others, and results, including customer, people, society, and key results. So far, the presented models are focused on the organization in a hierarchical way, but there are some others that advocate following a more horizontal approach, i.e., focused on the business processes. Two well-known examples are the framework proposed by Kueng (2000) for conceptualizing and developing a process performance measurement system and the Devil's quadrangle by Brand and Kolk (1995), which distinguishes four performance dimensions: time, cost, quality, and flexibility. These four dimensions have been established as the typical perspectives of business process performance measurement (Dumas et al. 2013; Jansen-Vullers et al. 2008).

PPI Definition Approaches

There is another research stream that has gained a great interest in PPM, mainly because the models mentioned above define different perspectives for which specific PPIs should be further defined but provide little guidance on how to define and operationalize them. Therefore, this research line is focused on the definition of PPIs, which has been addressed from different perspectives. In the formalization perspective, the work by Soffer and Wand (2005) proposes a set of formally defined concepts to enable the incorporation of measures and indicators (referred to as criterion functions for soft goals) into process modelling and design methods. Wetzstein et al. (2008), in their framework for business activity monitoring, describe a KPI ontology using the web service modelling language to specify PPIs over semantic business processes. Popova and Sharpanskykh (2010) formalize the definitions of PPIs as well

as their relationships with goals and among them using different variants of order-sorted predicate logic, which also enables formal analysis and verification. Finally, del Río-Ortega et al. (2013) present a metamodel for the definition of PPIs and its translation to a formal language (description logic) that allows a set of design-time analysis operations.

Regarding the provision of more user-friendly approaches for the definition of PPIs, while maintaining a formal foundation that allows further reasoning on those PPI definitions, several works can be named. Castellanos et al. (2005) propose the use of templates provided by a graphical user interface to define business measures related to process instances, processes, resources, or the overall business operations, which can be subject to posterior intelligent analysis to understand causes of undesired values and predict future values. del Río-Ortega et al. (2016) introduce a template-based approach that uses linguistic patterns for the definition of PPIs and is based on the PPINOT metamodel, thus enabling their automated computation and analysis. Saldivar et al. (2016) present a spreadsheet-based approach for business process analysis tasks that include writing metrics and assertions, running performance analysis and verification tasks, and reporting on the outcomes. Finally, van der Aa et al. (2017) use hidden Markov models to transform unstructured natural language descriptions into measurable PPIs.

Lastly, a set of researches have focused on developing graphical approaches for the definition of PPIs. Korherr and List (2007) extend two graphical notations for business process modelling (BPMN and EPC) to define business process and performance measures related to cost, quality, and cycle time. Friedenstab et al. (2012) extend BPMN for business activity monitoring by proposing a metamodel and a set of graphical symbols for the definition of PPIs, actions to be taken after their evaluation, and dashboards. del Río-Ortega et al. (2017b) present a graphical notation and editor to model PPIs together with business processes built on a set of principles for obtaining cognitively effective visual notations. The proposed notation is based on the PPINOT

metamodel and allows the automated computation of the PPI definitions.

PPI Analysis and Visualization Techniques

Concerning the visualization of process performance, researchers have focused on developing process-specific visual components that complement the use of general-purpose type of graphs such as bar graphs, bullet graphs, sparklines, or scatter plots in order to compare different cohorts of business process instances in order to spot process performance variations. Some of these visualizations include arranging and connecting pieces of information following the process model structure. For instance, in Wynn et al. (2017), a three-dimensional visualization of the process model enriched with a variety of performance metrics is used to identify differences between cohorts. Other visualizations adapt general-purpose graphs to visualize elements relevant to processes. Low et al. (2017) adapt social network graphs and timeline visualization approaches such as Gantt charts to highlight resource and timing changes; Claes et al. (2015) adapt dotted charts to represent the occurrence of events in time, and Richter and Seidl (2017) use a diagram similar to Gantt charts to represent temporal drifts in an event stream.

This last paper is also an example of a technique that analyses the performance data in order to obtain insights about process performance, in this case, temporal drifts. Similarly, other analysis techniques have been proposed such as in Hompes et al. (2017) in which the Granger causality test is used to discover causal factors that explain business process variation. Finally, process cubes can also be used during the analysis. Process cubes are based on the OLAP data cube concept. Each cell in the process cube corresponds to a set of events and can be used to compare and find process performance variation (Aalst 2013).

Examples of Application

Business process performance measurement is commonly applied for business process

control and continuous optimization. Process-oriented organizations use PPM as a tool to quantify issues or weaknesses associated to their business processes, like bottlenecks, and to identify improvement points. Sometimes, these organizations are engaged in initiatives to adopt sets of good practices or frameworks, either general ones, like the aforementioned EFQM, or specific to their domains, like ITIL or SCOR. In these cases, organizations apply the proposed PPM approach, according to the corresponding framework, where a series of performance measures to be defined are frequently proposed. The application of PPM can also be triggered by specific laws, related, for instance, to transparency or audits in the context, for example, of certifications. PPM can also help in deciding the viability of a business after a first short period of existence.

Future Directions for Research

While the definition of PPIs in the context of different performance measurement models has been widely analyzed and has produced a number of well-established approaches, it is recognized that they are suitable for structured business processes whose expected behavior is known *a priori*. Their direct application is, however, limited in the context of processes with a high level of variability or knowledge-intensive processes (KIPs), which are frequently non-repeatable, collaboration-oriented, unpredictable, and, in many cases, driven by implicit knowledge from participants, derived from their capabilities and previous experiences. In that context, new proposals pursue the incorporation of the new concepts involved in this type of processes, like the explicit knowledge used in process activities, into the definition of performance measures and indicators. Other research approaches focus on measuring the productivity of knowledge workers or their interaction and communication in KIPs. Nevertheless, a general proposal for the definition of performance measures in KIPs that can be used independently of the context and that addresses their particularities described above is missing.

A research area that is being recently explored is the development of techniques to enable evidence-driven performance diagnosis and improvement. Traditionally, they have been done based on the experience and intuition of process owners. However, nowadays, the amount and nature of available data produced by the information systems supporting business processes (e.g. event logs) make it possible for the application of different statistical and machine learning techniques for that purpose. Some preliminary approaches include a method for the identification of proper thresholds for a lead PPI in order to support the achievement of an associated lag PPI (del Río-Ortega et al. 2017a) or the use of statistical hypothesis testing to find root causes or performance drifts (Hompes et al. 2017).

Cross-References

- [Big Data Analysis for Social Good](#)
- [Business Process Analytics](#)
- [Business Process Event Logs and Visualization](#)
- [Business Process Model Matching](#)
- [Business Process Querying](#)
- [Predictive Business Process Monitoring](#)
- [Queue Mining](#)

References

- Aalst WMPvd (2013) Process cubes: slicing, dicing, rolling up and drilling down event data for process mining. In: Asia Pacific business process management (AP-BPM), pp 1–22
- Adams C, Neely A (2002) Prism reform. *Financ Manag* 5:28–31
- Brand N, Kolk H (1995) Workflow analysis and design. *Bedrijfswetenschappen*: Kluwer, Deventer
- Brignall T, Fitzgerald L, Johnston R, Silvestro R (1991) Performance measurement in service businesses. *Manag Account* 69(10):34–36
- Castellanos M, Casati F, Shan MC, Dayal U (2005) ibom: a platform for intelligent business operation management. In: Proceeding of the 21st international conference on data engineering, pp 1084–1095
- Claes J, Vanderfeesten I, Pinggera J, Reijers HA, Weber B, Poels G (2015) A visual analysis of the process of process modeling. *ISeB* 13(1):147–190
- del Río-Ortega A, García F, Resinas M, Weber E, Ruiz F, Ruiz-Cortés A (2017a) Enriching decision making with data-based thresholds of process-related kpis. In: Proceedings of 29th the international conference on advanced information systems engineering (CAiSE), pp 193–209
- del Río-Ortega A, Resinas M, Cabanillas C, Ruiz-Cortés A (2013) On the definition and design-time analysis of process performance indicators. *Inf Syst* 38(4): 470–490
- del Río-Ortega A, Resinas M, Durán A, Bernárdez B, Ruiz-Cortés A, Toro M (2017b) Visual PPINOT: a graphical notation for process performance indicators. *Bus Inf Syst Eng*. <https://link.springer.com/article/10.1007/s12599-017-0483-3>
- del Río-Ortega A, Resinas M, Durán A, Ruiz-Cortés A (2016) Using templates and linguistic patterns to define process performance indicators. *Enterp Inf Syst* 10(2):159–192
- Dumas M, La Rosa M, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer, Berlin/Heidelberg
- EFQM (2010) Efqm excellence model. Available from: <http://www.efqm.org/the-efqm-excellence-model>
- Few S (2006) Information dashboard design: the effective visual communication of data. O'Reilly Media, Inc., Beijing
- Friedenstab JP, Janiesch C, Matzner M (2012) Extending BPMN for business activity monitoring. In: Proceeding of the 45th Hawaii international conference on system science (HICSS), pp 4158–4167
- Hompes BFA, Maaradij A, Rosa ML, Dumas M, Buijs JCAM, Aalst WMPvd (2017) Discovering causal factors explaining business process performance variation. In: Advanced information systems engineering (CAiSE), pp 177–192
- Jansen-Vullers MH, Kleingeld PAM, Netjes M (2008) Quantifying the performance of workflows. *Inf Syst Manag* 25(4):332–343
- Kaplan RS, Norton DP (1996) The balanced scorecard: translating strategy into action. Harvard Business School Press, Boston
- Keagan D, Eiler R, Jones C (1989) Are your performance measures obsolete? *Manag Account* 70(12): 45–50
- Korherr B, List B (2007) Extending the EPC and the BPMN with business process goals and performance measures. In: Proceeding of the 9th international conference on enterprise information systems (ICEIS), vol 3, pp 287–294
- Kueng P (2000) Process performance measurement system: a tool to support process-based organizations. *Total Qual Manag* 11(1):67–85
- Low WZ, van der Aalst WMP, ter Hofstede AHM, Wynn MT, De Weerdt J (2017) Change visualisation: analysing the resource and timing differences between two event logs. *Inf Syst* 65:106–123
- Lynch R, Cross K (1991) Measure up!: the essential guide to measuring business performance. Mandarin, London

- McChesney C, Covey S, Huling J (2012) The 4 disciplines of execution: achieving your wildly important goals. Simon and Schuster, London
- Popova V, Sharpanskykh A (2010) Modeling organizational performance indicators. *Inf Syst* 35(4):505–527
- Richter F, Seidl T (2017) TESSERACT: time-drifts in event streams using series of evolving rolling averages of completion times. In: Business process management (BPM), pp 289–305
- Saldivar J, Vairetti C, Rodríguez c, Daniel F, Casati F, Alarcón R (2016) Analysis and improvement of business process models using spreadsheets. *Inf Syst* 57: 1–19
- Soffer P, Wand Y (2005) On the notion of soft-goals in business process modeling. *Bus Process Manag J* 11:663–679
- van der Aa H, Leopold H, del-Río-Ortega A, Resinas M, Reijers HA (2017) Transforming unstructured natural language descriptions into measurable process performance indicators using hidden markov models. *Inf Syst* 71:27–39
- Van Looy A, Shafagatova A (2016) Business process performance measurement: a structured literature review of indicators, measures and metrics. *SpringerPlus* 5(1797):1–24
- Wetzstein B, Ma Z, Leymann F (2008) Towards measuring key performance indicators of semantic business processes. *Bus Inf Syst* 7:227–238
- Wynn MT, Poppe E, Xu J, ter Hofstede AHM, Brown R, Pini A, van der Aalst WMP (2017) ProcessProfiler3d: a visualisation framework for log-based process performance comparison. *Decis Support Syst* 100:93–108

Business Process Querying

Artem Polyvyanyy
The University of Melbourne, Parkville, VIC,
Australia

Synonyms

[Process filtering](#); [Process management](#); [Process retrieval](#)

Definitions

Business process querying studies methods for managing, e.g., filtering and/or manipulating, repositories of models that describe observed

and/or envisioned business processes and relationships between the business processes.

A *process querying method* is a technique that given a process repository and a process query systematically implements the query in the repository, where a *process query* is a (formal) instruction to manage a process repository.

Overview

This encyclopedia entry summarizes the state-of-the-art methods for (business) process querying proposed in the publications included in the literature reviews reported in Wang et al. (2014); Polyvyanyy et al. (2017b) and the related works discussed in the reviewed publications. The scope of this entry is limited to the methods for process querying that are based on special-purpose (programming) languages for capturing process querying instructions and were developed in the areas of process mining (van der Aalst 2016) and business process management (Weske 2012; Dumas et al. 2013).

Next, this section lists basic concepts in process querying. Let \mathcal{U}_{an} and \mathcal{U}_{av} be the universe of *attribute names* and *attribute values*, respectively.

Event. An *event* e is a relation between some attribute names and attribute values with the property that each attribute name is related to exactly one attribute value, i.e., it is a partial function $e : \mathcal{U}_{an} \rightarrow \mathcal{U}_{av}$.

Process. A *process* is a partially ordered set $\pi := (E, \leq)$, where E is a set of events and \leq is a partial order over E .

Trace. A *trace* is a process $\pi := (E, <)$, where $<$ is a total order over E .

Behavior. A *behavior* is a multiset of processes.

Behavior model. A *behavior model* is a simplified representation of real-world or envisioned behaviors and relationships between the behaviors that serves a particular purpose for a target audience.

Process repository. A *process repository* is an organized collection of behavior models.

Process query. A *process query* is an instruction to filter or manipulate a process repository.

Let \mathcal{U}_{pr} and \mathcal{U}_{pq} be the universe of *process repositories* and the universe of *process queries*, respectively.

Process querying method. A *process querying method* m is a relation between ordered pairs, in which the first entries are process repositories and the second entries are process queries, and process repositories with the property that each pair is related to exactly one process repository, i.e., it is a function $m : \mathcal{U}_{pr} \times \mathcal{U}_{pq} \rightarrow \mathcal{U}_{pr}$.

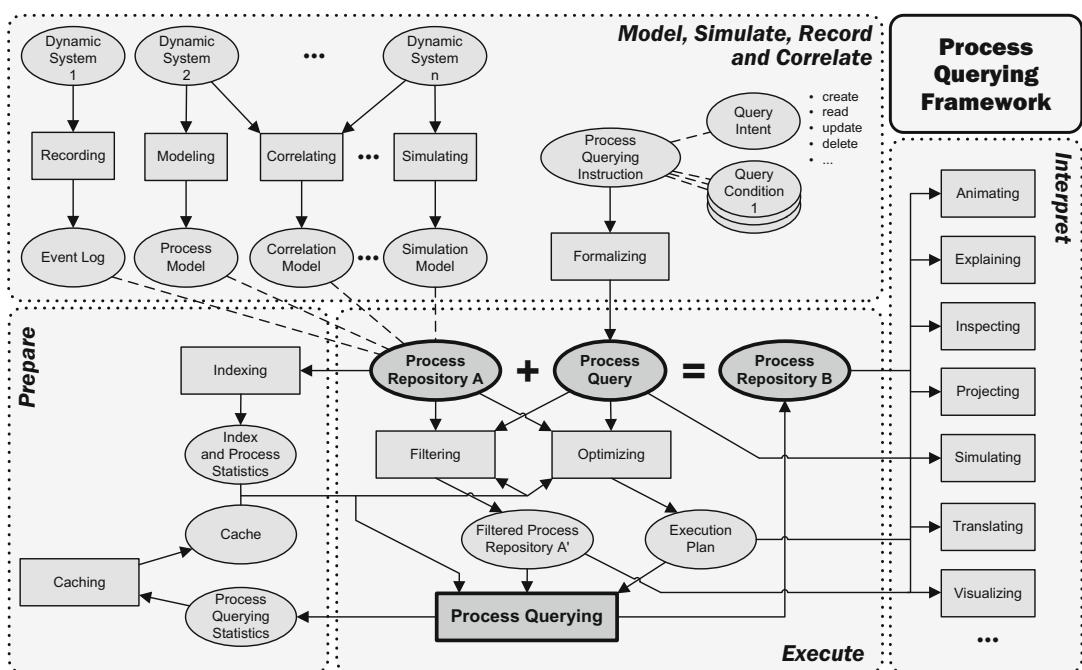
Framework

In Polyvyanyy et al. (2017b), a *process querying framework* is proposed. A schematic visualization of the framework is shown in Fig. 1.

The framework is an abstract system of components that provide generic functionality and can be selectively replaced to result in a

new process querying method. In the figure, rectangles and ovals denote *active components* and *passive components*, respectively. Active components represent actions performed by the process querying methods. Passive components stand for (collections of) data objects. The framework is composed of four parts. These parts are responsible for (i) designing process repositories and process queries, (ii) preparing and (iii) executing process queries, and (iv) interpreting results of the process querying methods; refer to Polyvyanyy et al. (2017b) for details.

The framework proposes four types of behavior models: event logs, process models, simulation models, and correlation models. An *event log* is a finite collection of traces, where each trace is a finite sequence of events observed and recorded in the real world. A process model is a conceptual model that describes behaviors. A simulation model is a process model with a part of its behavior. Finally, a correlation model is a model that describes relations between two behaviors. All the reported below process querying methods operate over event logs and process models.



Business Process Querying, Fig. 1 A schematic view of the process querying framework from (Polyvyanyy et al. 2017b)

Techniques

This section lists techniques that are often used as a basis for implementing process querying methods.

Model Analysis. Behavior models are often formalized as graphs. Several methods for process querying are grounded in the analysis of structural properties of graphs that encode behavior models, e.g., BP-QL, DMQL, GMQL, and VMQL. Examples of graph analysis tasks employed for process querying include the existence of a path problem and subgraph isomorphism problem.

SPARQL Protocol and RDF Query Language (SPARQL). SPARQL is a semantic query language for retrieving and manipulating data captured in Resource Description Framework (RDF) format (Hebeler et al. 2009). SPARQL allows users to write queries against data stored in the form of a set of “subject-predicate-object” triples.

Several methods for process querying are based on SPARQL, e.g., BPMN VQL and FPSPARQL. These methods encode process repositories as RDF data and implement process querying by first transforming process queries into SPARQL queries and then executing SPARQL queries over the RDF data.

Given a model of a finite-state system, for example, a behavior model, and a formal property, *model checking* techniques verify whether the property holds for the system (Baier and Katoen 2008).

Temporal Logic. A temporal logic is a formal system for encoding and reasoning about propositions qualified in terms of time (Øhrstrøm and Hasle 2007). Several methods for process querying are grounded in model checking temporal logic properties, e.g., BPMN-Q, BPSL, CRL, and PPSL. Given a process repository and process query, these methods proceed by translating the process query into a temporal logic expression, e.g., linear temporal logic (LTL), computation tree logic (CTL), and metrical temporal logic

(MTL). Then, each behavior model from the repository gets translated into a finite-state system and verified against the formula. Finally, behavior models that translate into systems for which the property captured in the formula holds constitute the result of the method.

First-Order Logic. First-order logic (FOL) is a formal logic that extends propositional logic which operates over declarative propositions with the use of predicates and quantifiers (Smullyan 1995). Several methods for process querying are grounded in model checking FOL properties, e.g., CRG and QuBPAL. These methods encode process repositories as formal FOL theories and implement process querying by verifying properties (encoded in process queries) of these systems.

Behavior Analysis. Several methods for process querying are grounded in the analysis of properties of behaviors encoded in models, e.g., APQL, BQL, and PQL. These methods support process queries that specify conditions over all the processes (of which there can be potentially infinitely many) encoded in the behaviors of models stored in a process repository.

Methods

This section discusses the state-of-the-art methods for process querying. The methods can be seen as instantiations of the process querying framework, refer to Fig. 1, with various concrete components. In what follows, the methods are grouped based on the types of behavior models that they can take as input in process repositories and are referred to by the names of the corresponding languages for specifying process queries.

Log Querying

Process querying methods proposed in this section operate over event logs.

CRG&eCRG. Compliance Rule Graph (CRG) is a visual language for specifying compliance

rules (Ly et al. 2010). The semantics of CRG is defined based on FOL over event logs. Extended Compliance Rule Graph (eCRG) extends CRG to allow modeling compliance rules over multiple process perspectives (Knuplesch and Reichert 2017). At this stage, eCRG addresses the interaction, time, data, and resource perspectives of process compliance. Similar to CRG, the execution semantics of eCRG is defined based on FOL over event logs.

DAPOQ-Lang. The Data-Aware Process Oriented Query Language (DAPOQ-Lang) allows querying over events in traces of event logs, as well as over related data objects, their versions and data schemas, and temporal properties of all the aforementioned elements (de Murillas et al. 2016).

FPSPARQL. The Folder-Path-enabled extension of SPARQL (FPSPARQL) is a query language founded on the two concepts of folders and paths that allow grouping events and exploring sequences of events recorded in event logs (Beheshti et al. 2011).

FPSPARQL operates over a data model that encodes an event log as a graph. In such a graph, nodes are events from the event log and edges stem from the orderings of events in the traces of the event log. Folders and paths are results of FPSPARQL queries. They constitute abstractions over the event log graphs. A folder abstraction refers to a collection of events, whereas a path abstraction encodes one or several paths in the transitive closure of the graph. Finally, FPSPARQL allows querying logs using standard SPARQL capabilities.

PIQL. Process Instance Query Language (PIQL) is an extension of Decision Model and Notation (DMN) for querying process performance indicators over executed traces (Pérez-Álvarez et al. 2016). DMN is a standard language for describing logic of decisions within organizations. PIQL is a textual language that resembles natural language. It can be used during process execution to define decision logic that

depends on the information kept in the so far executed traces.

Model Querying

This section lists methods for querying collections of process models. These methods can be split into those that operate over process model specifications and those that explore behaviors encoded in process models. Methods that operate over model specifications can be further split into two subclasses. The first subclass includes methods for querying conceptual models that were also reported useful for querying process models. The second subclass consists of dedicated techniques for querying process model collections.

DMQL, GMQL, and VMQL are the methods that belong to the first subclass of methods that operate over process model specifications.

DMQL&GMQL. The Generic Model Query Language (GMQL) is a textual query language designed to query collections of conceptual models created in arbitrary graph-based modeling languages (Delfmann et al. 2015b). GMQL querying is implemented via searching for model subgraphs that correspond to a predefined pattern query. GMQL is proposed to query process models, as well as data models, organizational charts, and other types of models.

The Diagrammed Model Query Language (DMQL) (Delfmann et al. 2015a) extends GMQL with a visual concrete syntax and support for additional requirements, e.g., ability to approximate analysis of the execution semantics of process models in the case of loops.

VMQL. The Visual Model Query Language (VMQL) is a language for querying collections of conceptual models (Störrle 2009, 2011). Among other types of conceptual models, the authors of VMQL demonstrated its use over collections of process models, e.g., Unified Modeling Language (UML) Activity Diagrams and Business Process Model and Notation (BPMN) models. VMQL is based on the concept of a host language. For example, VMQL for querying BPMN models subsumes the syntax and notation of BPMN. This approach allows decreasing the user effort

for reading and writing VMQL queries. VMQL queries can also specify constraints that go beyond the syntax of the host language.

For querying purposes, process models and VMQL queries are encoded in a Prolog database. Executing a VMQL query with respect to a given model requires finding parts of the model that resemble the structure and properties of the query and satisfy the constraints.

Next, we discuss dedicated process querying methods that operate over model specifications.

BPMN-Q. BPMN query language (BPMN-Q) was originally proposed to query business process definitions, i.e., (Awad 2007) the flow between activities, branching and joining structures, and data flow. BPMN-Q extends the abstract and concrete syntax of BPMN. Hence, BPMN-Q is a visual language. Later, BPMN-Q was extended to serve as a visual interface to temporal logics, e.g., LTL (Awad et al. 2008) and CTL (Awad et al. 2011).

BPMN VQL. BPMN Visual Query Language (BPMN VQL) is a visual query language (Francescomarino and Tonella 2008). The syntax and semantics of BPMN VQL are grounded in BPMN and SPARQL, respectively. The SPARQL translation of BPMN VQL queries is based on a formalization of the BPMN meta-model as an ontology. BPMN VQL queries consist of two parts. The matching part of a query specifies a structural condition to match in process models, whereas the selection part specifies parts of models to retrieve as query result.

BPSL. The Business Property Specification Language (BPSL) is a visual language that aims to facilitate the reasoning over business process models (Liu et al. 2007). The authors proposed direct translations of BPSL queries in both LTL and CTL to simplify the integration of BPSL with the existing formal verification tools. The language proposes dedicated operators for the recurring patterns in regulatory process compliance and specifies extension points for the definition and reuse of domain-specific temporal properties.

CRL. Compliance Request Language (CRL) is a visual language that is grounded in temporal logic and encompasses a range of compliance patterns (Elgammal et al. 2016). At this stage, CRL supports compliance patterns that address control flow, resource, and temporal aspects of business processes. During execution, control flow and resource patterns get mapped to LTL, whereas temporal patterns are translated into MTL formulas. MTL extends LTL and is interpreted over a discrete time domain using the digital clock model.

Descriptive PQL. Descriptive Process Query Language (Descriptive PQL) is a textual query language for retrieving process models, specifying abstractions on process models, and defining process model changes (Kammerer et al. 2015). The aforementioned operations are grounded in manipulations with single-entry-single-exit subgraphs (Polyvyanyy et al. 2010) of the process models.

IPM-PQL. Integrated Process Management-Process Query Language (IPM-PQL) is an XML-based language to query process model collections based on structure matching (Choi et al. 2007). IPM-PQL queries follow the structure of SQL queries and support four types of query conditions: process-has-attribute, process-has-activity, process-has-subprocess, and process-has-transition. When executed, IPM-PQL queries are transformed into XML Query (XQuery) expressions.

PPSL. The Process Pattern Specification Language (PPSL) is a lightweight extension of the language for capturing UML activity diagrams that allows users to model process constraints, e.g., those related to quality management (Förster et al. 2005). The semantics of the language is given as translations of PPSL queries into temporal logic, viz., LTL formulas (Förster et al. 2007).

Finally, we summarize process querying methods that operate over behaviors encoded in models.

APQL. A Process-model Query Language (APQL) is a query language based on semantic relations between occurrences of tasks specified in process models (ter Hofstede et al. 2013). The language allows composing the relations between tasks to obtain complex queries. APQL is precise, i.e., its semantics is defined over all possible execution traces of process models. The language is independent of any particular notation used to specify process models.

BQL. Business Query Language (BQL) is a textual query language based on semantic relations between occurrences of tasks specified in process models (Jin et al. 2011). Given a repository of process models, BQL can be used to retrieve the models satisfying the requirements based on the behavior encoded in the models. To compute the relations, BQL relies on the technology of complete prefix unfolding (McMillan 1995).

QuBPAL. Querying Business Process Abstract modeling Language (QuBPAL) is an ontology-based language for the retrieval of process fragments to be reused in the composition of new business processes (Smith et al. 2010). Using QuBPAL, the user can query over three process perspectives: the structure of process specifications, the behavioral semantics of process specifications, and the pre-modeled domain knowledge about the business scenario encoded in models. QuBPAL queries are evaluated over Business Process Knowledge Base—a collection of FOL theories that formalize the process repository of semantically enriched process models.

PQL. Process Query Language (PQL) is a textual query language based on semantic relations between tasks in process models (Polyvyanyy et al. 2015). PQL extends the abstract syntax of APQL and has an SQL-like concrete syntax. The semantic relations between tasks are computed as aggregations over all the possible occurrences of these tasks, as specified in the behavior of the corresponding process model.

PQL is based on the semantic relations of the 4C spectrum (Polyvyanyy et al. 2014) – a sys-

tematic collection of the co-occurrence, conflict, causality, and concurrency relations. In addition to filtering capabilities, PQL addresses manipulation of process models. For example, PQL queries can specify instructions to insert traces into process models, i.e., to augment process models to additionally describe fresh traces. PQL implements instructions to insert new traces into process models as solutions to the process model repair problem (Polyvyanyy et al. 2017a).

Log and Model Querying

Methods listed below were proposed to query over collections that may include process models and/or event logs.

BPQL. Business Process Query Language (BPQL) is a textual language for querying over process models and event logs (Momotko and Subieta 2004). The language is defined as an extension of the Stack-Based Query Language (SBQL) (Subieta 2009). The semantics of the language is defined over the proposed abstract model for capturing process specifications and execution traces. The authors of BPQL proposed BPQL templates to use for monitoring process execution and integrated the language with the XML Process Definition Language (XPDL).

BP-QL. Business Process Query Language (BPQL) is a visual language that allows retrieving paths in process specifications (Beeri et al. 2006). BPQL operates over a dedicated abstract model for representing collections of processes. This model is used to distinguish queries that can be efficiently computed, queries that have high computation costs, and queries that cannot be computed at all. When a BPQL query is evaluated, its structural patterns are matched against process specifications. BPQL uses the proposed abstract model to support queries that can be answered in time polynomial in the size of a process specification. This is achieved at a price of precision of query answers. BPQL supports mechanisms for constructing concise finite representations of the (possibly infinite) query results. BP-Mon and BP-Ex are the two extensions of BPQL to support process

monitoring and querying of executed traces, respectively.

Applications

Due to their generic purpose, i.e., manipulation and filtering of process repositories, process querying methods have broad application in process mining and business process management. The aforementioned process querying methods were proposed and/or successfully applied to solve various problems in these fields. Some examples of the addressed problems include: business process compliance management, business process weakness detection, process variance management, process model translation, syntactical correctness checking, process model comparison, infrequent behavior detection, process instance migration, process monitoring, process reuse, and process standardization.

Future Directions for Research

Future work on process querying will contribute to achieving *process querying compromise* (Polyvyanyy et al. 2017b), i.e., will propose new and improve the existing process querying methods with the support of efficiently computable and practically relevant process queries. As of today, the existing works on process querying constitute non-coordinated efforts. Future work will contribute to the consolidation of various process querying methods leading to the definition of a standard meta-model for behavior models and behaviors that these models describe and a standard language for capturing process queries.

Most existing methods for process querying operate over specifications of behavior models. Future work will lead to new methods for querying based on behaviors encoded in models and study their practical implications.

While there is a plethora of methods and languages for capturing instructions for filtering process repositories, methods for manipulating process repositories are still in their infancy.

Cross-References

- [Business Process Model Matching](#)
- [Graph Pattern Matching](#)
- [Graph Query Languages](#)

References

- Awad A (2007) BPMN-Q: a language to query business processes. In: Enterprise modelling and information systems architectures, GI. Lecture notes in informatics, vol P-119. pp 115–128
- Awad A, Decker G, Weske M (2008) Efficient compliance checking using BPMN-Q and temporal logic. In: Business process management. Lecture notes in computer science, vol 5240. Springer, Berlin, pp 326–341. https://doi.org/10.1007/978-3-540-85758-7_24
- Awad A, Weidlich M, Weske M (2011) Visually specifying compliance rules and explaining their violations for business processes. J Vis Lang Comput 22(1):30–55. <https://doi.org/10.1016/j.jvlc.2010.11.002>
- Baier C, Katoen J (2008) Principles of model checking. MIT Press, Cambridge
- Beeri C, Eyal A, Kamenkovich S, Milo T (2006) Querying business processes. In: Very large data bases. ACM, pp 343–354
- Beheshti S, Benatallah B, Nezhad HRM, Sakr S (2011) A query language for analyzing business processes execution. In: Business process management. Lecture notes in computer science, vol 6896. Springer, Berlin/Heidelberg, pp 281–297. https://doi.org/10.1007/978-3-642-23059-2_22
- Choi I, Kim K, Jang M (2007) An XML-based process repository and process query language for integrated process management. Knowl Process Manag 14(4):303–316. <https://doi.org/10.1002/kpm.290>
- de Murillas EGL, Reijers HA, van der Aalst WMP (2016) Everything you always wanted to know about your process, but did not know how to ask. In: Business process management workshops: process querying. Lecture notes in business information processing, vol 281. pp 296–309. https://doi.org/10.1007/978-3-319-58457-7_22
- Delfmann P, Breuker D, Matzner M, Becker J (2015a) Supporting information systems analysis through conceptual model query – the diagramed model query language (DMQL). Commun Assoc Inf Syst 37:24
- Delfmann P, Steinhorst M, Dietrich H, Becker J (2015b) The generic model query language GMQL – conceptual specification, implementation, and runtime evaluation. Inf Syst 47:129–177. <https://doi.org/10.1016/j.is.2014.06.003>
- Dumas M, Rosa ML, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer, Berlin/Heidelberg. <https://doi.org/10.1007/978-3-642-33143-5>

- Elgammal A, Turetken O, Heuvel WJ, Papazoglou M (2016) Formalizing and applying compliance patterns for business process compliance. *Softw Syst Model* 15(1):119–146. <https://doi.org/10.1007/s10270-014-0395-3>
- Förster A, Engels G, Schattkowsky T (2005) Activity diagram patterns for modeling quality constraints in business processes. In: Model driven engineering languages and systems. Lecture notes in computer science, vol 3713. Springer, pp 2–16. https://doi.org/10.1007/11557432_2
- Förster A, Engels G, Schattkowsky T, Straeten RVD (2007) Verification of business process quality constraints based on visual process patterns. In: Theoretical Aspects of Software Engineering. IEEE Computer Society, pp 197–208. <https://doi.org/10.1109/TASE.2007.56>
- Francescomarino CD, Tonella P (2008) Crosscutting concern documentation by visual query of business processes. In: Business process management. Lecture notes in business information processing, vol 17. Springer, Berlin/Heidelberg, pp 18–31. https://doi.org/10.1007/978-3-642-00328-8_3
- Hebeler J, Ryan B, Andrew Perez-Lopez MF, Dean M (2009) Semantic web programming, 1st edn. Wiley, Indianapolis
- Jin T, Wang J, Wen L (2011) Querying business process models based on semantics. In: Database systems for advanced applications. Lecture notes in computer science, vol 6588. Springer, Berlin/Heidelberg, pp 164–178. https://doi.org/10.1007/978-3-642-20152-3_13
- Kammerer K, Kolb J, Reichert M (2015) PQL – A descriptive language for querying, abstracting and changing process models. In: Enterprise, business-process and information systems modeling. Lecture notes in business information processing, vol 214. Springer, pp 135–150. https://doi.org/10.1007/978-3-319-19237-6_9
- Knuplesch D, Reichert M (2017) A visual language for modeling multiple perspectives of business process compliance rules. *Softw Syst Model* 16(3):715–736. <https://doi.org/10.1007/s10270-016-0526-0>
- Liu Y, Müller S, Xu K (2007) A static compliance-checking framework for business process models. *IBM Syst J* 46(2):335–362. <https://doi.org/10.1147/sj.462.0335>
- Ly LT, Rinderle-Ma S, Dadam P (2010) Design and verification of instantiable compliance rule graphs in process-aware information systems. In: Advanced information systems engineering. Lecture notes in computer science, vol 6051. Springer, Berlin/Heidelberg, pp 9–23. https://doi.org/10.1007/978-3-642-13094-6_3
- McMillan KL (1995) A technique of state space search based on unfolding. *Formal Methods Syst Des* 6(1):45–65. <https://doi.org/10.1007/BF01384314>
- Momotko M, Subieta K (2004) Process query language: a way to make workflow processes more flexible. In: Advances in databases and information systems. Lecture notes in computer science, vol 3255. Springer, Berlin/Heidelberg, pp 306–321. https://doi.org/10.1007/978-3-540-30204-9_21
- Øhrstrøm P, Hasle P (2007) Temporal Logic: from ancient ideas to artificial intelligence. Studies in linguistics and philosophy. Springer, Dordrecht
- Pérez-Álvarez JM, López MTG, Parody L, Gasca RM (2016) Process instance query language to include process performance indicators in DMN. In: IEEE enterprise distributed object computing workshops. IEEE Computer Society, pp 1–8. <https://doi.org/10.1109/EDOCW.2016.7584381>
- Polyvyany A, Vanhalto J, Völzer H (2010) Simplified computation and generalization of the refined process structure tree. In: Web services and formal methods. Lecture notes in computer science, vol 6551. Springer, pp 25–41. https://doi.org/10.1007/978-3-642-19589-1_2
- Polyvyany A, Weidlich M, Conforti R, Rosa ML, ter Hofstede AHM (2014) The 4C spectrum of fundamental behavioral relations for concurrent systems. In: Application and theory of petri nets and concurrency. Lecture notes in computer science, vol 8489. Springer, pp 210–232. https://doi.org/10.1007/978-3-319-07734-5_12
- Polyvyany A, Corno L, Conforti R, Raboczi S, Rosa ML, Fortino G (2015) Process querying in Apromore. In: Business process management demo session, CEUR-WS.org, CEUR workshop proceedings, vol 1418, pp 105–109
- Polyvyany A, van der Aalst WMP, ter Hofstede AHM, Wynn MT (2017a) Impact-driven process model repair. *ACM Trans Softw Eng Method* 25(4):1–60. <https://doi.org/10.1145/2980764>
- Polyvyany A, Ouyang C, Barros A, van der Aalst WMP (2017b) Process querying: enabling business intelligence through query-based process analytics. *Decis Support Syst* 100:41–56. <https://doi.org/10.1016/j.dss.2017.04.011>
- Smith F, Missikoff M, Proietti M (2010) Ontology-based querying of composite services. In: Business system management and engineering. Lecture notes in computer science, vol 7350. Springer, Berlin/Heidelberg, pp 159–180. https://doi.org/10.1007/978-3-642-32439-0_10
- Smullyan RM (1995) First-order logic. In: Dover (ed) Courier Corporation, Inc. New York
- Störrle H (2009) VMQL: A generic visual model query language. In: IEEE Visual languages and human-centric computing, IEEE computer society, pp 199–206. <https://doi.org/10.1109/VLHCC.2009.5295261>
- Störrle H (2011) VMQL: a visual language for ad-hoc model querying. *J Vis Lang Comput* 22(1):3–29. <https://doi.org/10.1016/j.jvlc.2010.11.004>
- ter Hofstede AHM, Ouyang C, Rosa ML, Song L, Wang J, Polyvyany A (2013) APQL: a process-model query language. In: Asia Pacific business process management. Lecture notes in business information processing, vol 159. Springer, pp 23–38. https://doi.org/10.1007/978-3-319-02922-1_2

- van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer, Berlin/Heidelberg. <https://doi.org/10.1007/978-3-662-49851-4>
- Subieta K (2009) Stack-based query language. In: Liu L (ed) Encyclopedia of database systems, Springer, New York, pp 2771–2772. https://doi.org/10.1007/978-0-387-39940-9_1115
- Wang J, Jin T, Wong RK, Wen L (2014) Querying business process model repositories – a survey of current approaches and issues. *World Wide Web* 17(3):427–454. <https://doi.org/10.1007/s11280-013-0210-z>
- Weske M (2012) Business process management – concepts, languages, architectures, 2nd edn. Springer, Berlin/Heidelberg. <https://doi.org/10.1007/978-3-642-28616-2>
-

Business Process Variants Analysis

- Business Process Deviance Mining

C

Caching for SQL-on-Hadoop

Gene Pang and Haoyuan Li
Alluxio Inc., San Mateo, CA, USA

Definitions

Caching for SQL-on-Hadoop are techniques and systems which store data to provide faster access to that data, for Structured Query Language (SQL) engines running on the Apache Hadoop ecosystem.

Overview

The Apache Hadoop software project (Apache Hadoop 2018) has grown in popularity for distributed computing and big data. The Hadoop stack is widely used for storing large amounts of data, and for large-scale, distributed, and fault-tolerant data processing of that data. The Hadoop ecosystem has been important for organizations to extract actionable insight from the large volumes of collected data, which is difficult or infeasible for traditional data processing methods.

The main storage system for Hadoop is the Hadoop Distributed File System (HDFS). It is a distributed storage system which provides fault-tolerant and scalable storage. The main data processing framework for Hadoop is MapReduce, which is based on the Google MapReduce project

(Dean and Ghemawat 2008). MapReduce is a programming model and distributed batch processing framework for reliably processing large volumes of data, typically from HDFS. However, the distinct programming model for MapReduce can be a barrier for nondevelopers, such as data analysts or business intelligence (BI) tools.

Because of this barrier to entry, new tools and frameworks have emerged for the Hadoop ecosystem. Primarily, new Structured Query Language (SQL) frameworks have gained popularity for the Hadoop software stack. SQL is a domain-specific declarative language to describe retrieving and manipulating data, typically from relational database management systems (RDBMS). These SQL engines for Hadoop have been effective for empowering more users who are familiar with existing tools and the SQL language for managing data. These frameworks typically read data from HDFS, process the data in a distributed way, and return to the user the desired answer. Many SQL engines are available for Hadoop, which include Apache Hive (2018), Apache Spark SQL (2018), Apache Impala (2018), Presto (Facebook 2018), and Apache Drill (2018).

These SQL on Hadoop engines have combined the convenience of data query with SQL and the power of distributed processing of data with Hadoop. However, these engines are commonly not tightly integrated into the Hadoop ecosystem. There can be advantages of separating the computation engine from the storage system, such as cost effectiveness and operational flexibility.

However, one potential weakness is that the performance of accessing data may decline. Since RDBMS typically have tight integration and control of the entire stack from the computation to the storage, performance may be faster and more predictable. For comparable experiences with traditional SQL processing in RDBMS, distributed SQL engines on Hadoop turn to caching to achieve the desired performance.

There are several different ways SQL engines on Hadoop can take advantage of caching to improve performance. The primary methods are as follows:

- Internal caching within the SQL engine
- Utilizing external storage systems for caching

SQL Engine Internal Caching

A common way a SQL engine uses caching is to implement its own internal caching. This provides the most control for each SQL engine as to what data to cache and how to represent the cached data. This internal cache is also most likely the fastest to access, since it is located closest to the computation being performed (same memory address space or local storage), in the desired format. However, the SQL engine internal caching may not be shareable between different users and queries, which is beneficial in a multi-tenant environment. Also, not all SQL processing frameworks have implemented an internal cache, which prevents using cached data for queries.

Apache Spark SQL

Apache Spark SQL is a SQL query engine built on top of Apache Spark, a distributed data processing framework. Spark SQL exposes a concept called a DataFrame, which represents a distributed collection of structured data, similar to a table in an RDBMS. Caching in Spark SQL utilizes the caching in the Spark computation engine. A user can choose to cache a DataFrame in Spark SQL by explicitly invoking a cache command (Spark SQL 2018). When the command is invoked, Spark SQL engine will cache the DataFrame internal to the Spark engine.

There are several user-configurable ways the data can be cached, including MEMORY_ONLY, MEMORY_AND_DISK, DISK_ONLY, and others (Spark RDD 2018). An explicit command must be invoked to remove the DataFrame from the cache.

Apache Hive and LLAP

Apache Hive is a SQL data warehouse for Hadoop, which utilizes an existing distributed computation engine for the data processing. The options of computation engines for Hive to use are MapReduce, Spark, or Tez. Hive uses Live Long And Process (LLAP) (Apache Hive LLAP 2018) for caching data for SQL processing. Currently, only Tez will take advantage of the caching available via LLAP.

LLAP is a long-lived daemon which runs alongside Hadoop components in order to provide caching and data pre-fetching benefits. Since Tez takes advantage of LLAP, when Hive uses Tez as the computation engine, data access will go to the LLAP daemons, instead of the HDFS DataNodes. With this access pattern and architecture, LLAP daemons can cache data for Hive queries.

External Storage Systems for Caching

Another major technique for caching for SQL engines on Hadoop is to utilize an external storage system for caching data. Using a separate system can have several benefits. An external system may be able to manage the cache more effectively and provide additional cache-related features. Also, an external system can be deployed independently from the other components of the ecosystem, which can provide operational flexibility. Additionally, sharing cached data can be enabled or made simpler with a separate system for caching. By using a separate system for handling the caching of data, SQL engines can take advantage by reading the data from the cache, instead of the originating data source. This can greatly improve performance, especially when frequently accessed files are cached. Below

are some of the main systems which can provide caching in Hadoop ecosystems.

Alluxio

Alluxio (2018) is a memory speed virtual distributed storage system. Alluxio provides a unified namespace across multiple disparate storage systems. Alluxio enables users to “mount” any existing storage system like HDFS, Amazon S3, or a storage appliance, and presents a single, unified namespace encompassing all the data. When a mount is established, applications simply interact with the Alluxio namespace, and the data will be transparently accessed from the mounted storage and cached in Alluxio. Since there is no limit to how many mounts are possible, Alluxio enables accessing all data from a single namespace and interface, and allows queries to span multiple different data sources seamlessly. Figure 1 shows how Alluxio can mount multiple storage systems for multiple computation frameworks.

Alluxio enables caching for SQL engines by providing memory speed access to data via its memory centric architecture. Alluxio stores data on different tiers, which include memory, solid-state drive (SSD), and disk. With the memory tier and the tiered storage architecture, Alluxio can store the important and most frequently accessed

data in the fastest memory tier, and store less accessed data in slower tiers with more capacity.

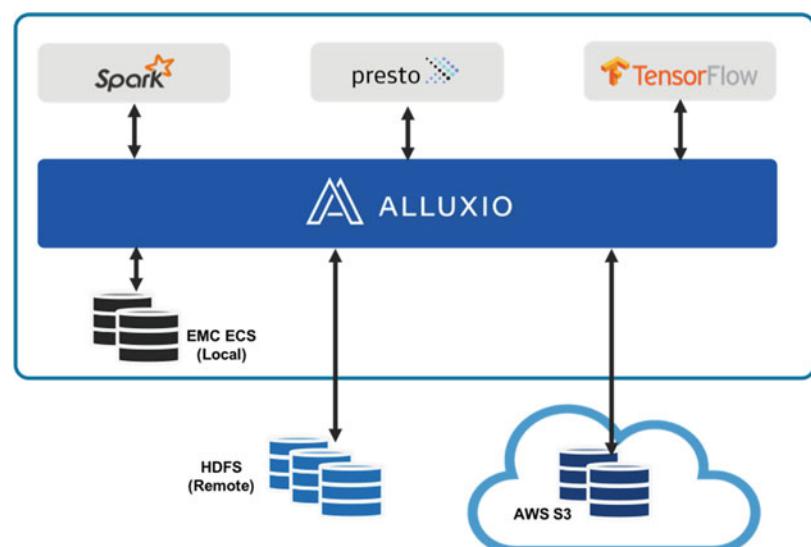
SQL engines can access any data via the Alluxio unified namespace, and the data will be either fetched from the configured mounted storages, or served from Alluxio-managed storage (memory, SSD, and disk), which can greatly improve performance. When Alluxio is deployed colocated with the computation cluster, the Alluxio memory storage can behave similar to the internal caches of SQL engines. Alluxio is able to transparently cache data from various other storage systems, like HDFS. Additionally, since the Alluxio storage is independent from the SQL engines, the cached data can easily be shared between different computation frameworks, thus allowing greater flexibility for deployments.

Apache Ignite HDFS Cache

Apache Ignite (2018) is a distributed platform for in-memory storage and computation. Apache Ignite is a full stack of components which includes a key/value store, a SQL processing engine, and a distributed computation framework, but can be used to cache data for HDFS and SQL engines on Hadoop.

Apache Ignite provides a caching layer for HDFS via the Ignite File System (IGFS). All clients and applications interact with IGFS for data, and IGFS handles reading and writing the

Caching for SQL-on-Hadoop, Fig. 1
Alluxio mounting various storage systems



data to memory on the nodes. When using IGFS as a cache, users can optionally configure a secondary file system for IGFS, which allows transparent read-through and write-through behavior. Users can configure HDFS for the secondary file system, so when the data is synced to IGFS, the data in Ignite is cached in memory.

Using Apache Ignite, SQL engines can access their HDFS data via IGFS, and the data can be cached in the Ignite memory storage. Any SQL engine can take advantage of this cached data for greater performance for queries. When using IGFS, the SQL engine internal caching does not need to be used, which will also enable sharing of the cached data. However, IGFS only allows a single secondary HDFS-compatible file system, if SQL queries need to access varied sources, IGFS would not be able to cache the data.

HDFS Centralized Cache Management

Another external caching option is for the cache to be implemented in the Hadoop storage system, HDFS. Depending on the operating system (OS) buffer cache is one way to take advantage of caching in HDFS. However, there is no user control over the OS buffer cache. Instead, HDFS has a centralized cache management feature (Apache Hadoop HDFS 2018), which enables users to explicitly specify files or directories which should be cached in memory. Once a user specifies a path to cache in HDFS, the HDFS DataNodes will be notified, and will cache the data in off-heap memory. In order to remove a file or directory from the cache, a separate command must be invoked. HDFS caching can help accelerate access to the data, since the data can be stored in memory, instead of disk.

By enabling the HDFS centralized cache, the data of specified files will reside on memory on the HDFS DataNodes, so any SQL engines or applications accessing those files will be able to read the data from memory. Increasing the data access performance of HDFS will help any queries accessing the cached files. Apache Impala has additional support for utilizing the HDFS centralized caching feature for Impala tables

which are read from HDFS. With Apache Impala, users can specify tables or partitions of tables to be cached via HDFS centralized caching. IBM Big SQL (Floratou et al. 2016) also uses the HDFS cache for accelerating access to data and utilizes new caching algorithms Adaptive SLRU-K and Adaptive EXD to improve the cache hit rates for better performance.

Since the centralized caching is an HDFS feature, it can help improve performance for any HDFS data access. However, for disaggregated clusters, where the HDFS cluster is separate from the SQL engine cluster, the HDFS caching may not be able to help much. Separating computation and storage clusters is becoming popular for its scalability, flexibility, and cost effectiveness. In these environments, when the computation cluster requires data, the data must be accessed across the network, so even if the data resides in the memory cache on the HDFS DataNodes, the network transfer may be the performance bottleneck. Therefore, further improving performance via caching requires either SQL engine internal caching or a separate caching system. Also, since the caching system is tightly integrated with HDFS, the deployment is not as flexible as other external caching systems.

Conclusion

Caching can significantly improve query performance for SQL engines on Hadoop. There are several different techniques for caching data for SQL frameworks. SQL engine internal caching has the potential for the greatest performance, but each framework must implement a cache, and the cached data is not always shareable by other applications or queries. Another option is to use the HDFS centralized caching feature to store files in memory on the HDFS DataNodes. This server-side cache is useful to speed up local access to data, but if SQL engines are not colocated with HDFS, or the data is not stored in HDFS, the network may become the performance bottleneck. Finally, external systems can cache data for SQL frameworks on Hadoop. When colocated with the computation cluster, external systems can store

data in memory similar to an internal cache, but provide the flexibility to share the data effectively between different applications or queries.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Hadoop](#)
- ▶ [Hive](#)
- ▶ [Spark SQL](#)
- ▶ [Virtual Distributed File System: Alluxio](#)

References

- Alluxio (2018) Alluxio – open source memory speed virtual distributed storage. <https://www.alluxio.org/>. Accessed 19 Mar 2018
- Apache Drill (2018) Apache Drill. <https://drill.apache.org>. Accessed 19 Mar 2018
- Apache Hadoop (2018) Welcome to Apache Hadoop! <http://hadoop.apache.org>. Accessed 19 Mar 2018
- Apache Hadoop HDFS (2018) Centralized cache management in HDFS. <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Centralized-CacheManagement.html>. Accessed 19 Mar 2018
- Apache Hive (2018) Apache Hive. <https://hive.apache.org>. Accessed 19 Mar 2018
- Apache Hive LLAP (2018) LLAP. <https://cwiki.apache.org/confluence/display/Hive/LLAP>. Accessed 19 Mar 2018
- Apache Ignite (2018) Apache Ignite. <https://ignite.apache.org/index.html>. Accessed 19 Mar 2018
- Apache Impala (2018) Apache Impala. <https://impala.apache.org>. Accessed 19 Mar 2018
- Apache Spark SQL (2018) Spark SQL. <https://spark.apache.org/sql/>. Accessed 19 Mar 2018
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
- Facebook (2018) Presto. <https://prestodb.io>. Accessed 19 Mar 2018
- Floratou A et al (2016) Adaptive caching in big SQL using the HDFS cache. In: SoCC’16 proceedings of the seventh ACM symposium on cloud computing, Snata Clara, 5–7 Oct 2016
- Spark RDD (2018) RDD programming guide. <http://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence>. Accessed 19 Mar 2018
- Spark SQL (2018) Spark SQL, dataframes and datasets guide. <http://spark.apache.org/docs/latest/sql-programming-guide.html#caching-data-in-memory>. Accessed 19 Mar 2018

Cartography

- ▶ [Visualization](#)

C

Causal Consistency

- ▶ [TARDiS: A Branch-and-Merge Approach to Weak Consistency](#)

Certification

- ▶ [Auditing](#)

Cheap Data Analytics on Cold Storage

Raja Appuswamy

Data Science Department, EURECOM, Biot, France

Definitions

Driven by the desire to extract insights out of data, businesses have started aggregating vast amounts of data from diverse data sources. However, recent analyst reports claim that only 10–20% of data stored is actively accessed with the remaining 80% being *cold* or infrequently accessed. Cold data has also been identified as the fastest-growing storage segment, with a 60% cumulative annual growth rate (Mendoza 2013; Moore 2015; Nandkarni 2014).

As the amount of cold data increases, enterprise customers are increasingly looking for more cost-efficient ways to store this data. A recent report from IDC emphasized the need for such low-cost storage by stating that only 0.5% of potential Big Data is being analyzed, and in order to benefit from unrealized value extraction, infrastructure support is needed to store large volumes of data, over long time duration, at extremely low cost

(Nandkarni 2014). Thus, the topic of managing cold data has received a lot of attention from both industry and academia over the past few years. Systems that are purpose-built to provide cost-efficient storage and analysis of cold data are referred to as *cold storage* systems.

Background and Overview

Enterprise databases have long used storage tiering for reducing capital and operational expenses. Traditionally, databases used a two-tier storage hierarchy. An *online* tier based on enterprise *hard disk drives (HDD)* provided low-latency (ms) access to data. The *backup* tier based on offline tape cartridges or optical drives, in contrast, provided low-cost, high-latency (hours or days) storage for storing backups to be restored only during rare failures.

As databases grew in popularity, the necessity to reduce recovery time after failure became important. Further, as regulatory compliance requirements forced enterprises to maintain long-term data archives, the offline nature of the backup tier proved too slow for both storing and retrieving infrequently accessed archival data. This led to the emergence of a new *archival* tier based on nearline storage devices, like virtual tape libraries (VTL), which could store and retrieve data automatically without human intervention in minutes.

Cheap Data Analytics on Cold Storage, Fig. 1

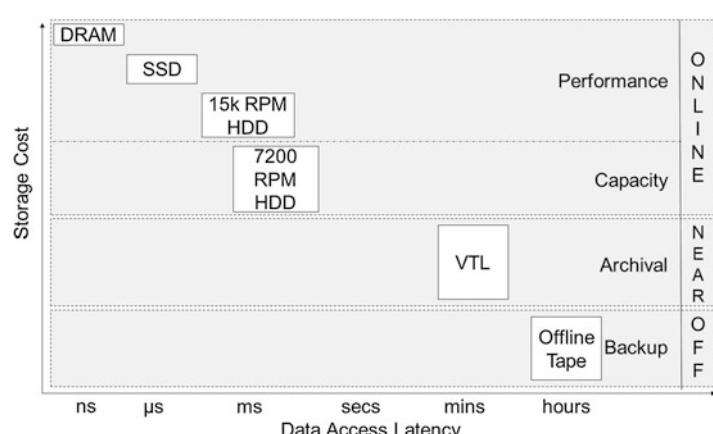
Storage tiering for enterprise databases

Over the past decade, the emergence of flash-based solid-state storage, declining price of DRAM, and demand for low-latency, real-time data analytics have resulted in the traditional online tier being bifurcated into two subtiers, namely, a *performance* tier based on RAM/SSD and a *capacity* tier based on HDD. Thus, most modern enterprises today use a four-tier storage hierarchy (performance, capacity, archival, backup) implemented using three storage types (online, nearline, and offline) as shown in Fig. 1.

Given the proliferation of cold data over the past few years, the obvious question that arises is where should such cold data be stored. Given that databases already have a tiered storage infrastructure in place, an obvious low-cost solution to deal with this problem is to store cold data in either the capacity tier or the archival tier. Unfortunately, both these options are not ideal given recent trends in the high-density storage hardware landscape.

HDD-Based High-Density Storage

Traditionally, 7,200 RPM HDDs have been the primary storage media used for provisioning the capacity tier. For several years, areal density improvements enabled HDDs to increase capacity at Kryder's rate (40% per year), outstripping the 18-month doubling of transistor count predicted by Moore's law. However, over the past few years, HDD vendors have hit walls in scaling areal density with conventional perpendicular magnetic recording (PMR) techniques (Fontana and Decad



2015). As a result, HDD vendors have started working on several techniques to improve areal density, like using helium-bearing instead of air-bearing for the disk head to pack more platters tightly or using shingled magnetic recording (SMR) techniques that pack more data by overlapping adjacent disk tracks, to further increase areal density. Despite such attempts from HDD vendors, HDD areal density is improving only at an annual rate of 16% instead of 40% (Fontana and Decad 2015; Moore 2016).

HDDs also present another problem when used as the storage medium of choice for storing cold data, namely, high idle power consumption. Unlike tape drives, which consume no power once unmounted, HDDs consume a substantial amount of power even while idle (Colarelli and Grunwald 2002). Such power consumption translates to a proportional increase in operational expenses. Thus, the capacity tier, with its always-on HDD-based storage, is not an ideal option for storing infrequently accessed cold data.

Tape-Based High-Density Storage

Unlike HDDs, the areal density of tapes has been increasing steadily at a rate of 33% per year, and the *Linear Tape Organization (LTO)* roadmap (LTO Ultrium 2015) projects continued increase in density for the foreseeable future. Today, a single LTO-7 cartridge is capable of matching or even outperforming a HDD, with respect to sequential data access bandwidth. As modern tape libraries use multiple drives, the cumulative bandwidth achievable using even low-end tape libraries is 1–2 GB/s. However, the random access latency of tape libraries is still 10,000× higher than HDDs (minutes versus ms) due to the fact that tape libraries need to mechanically load tape cartridges before data can be accessed. Thus, tape-based archival tier is used to store only rarely accessed compliance and backup data. As the expected workload in such cases is dominated by sequential writes with a few rare reads, the high access latency of tape drives is tolerable.

Using the archival tier to store cold data, however, changes the application workload, as analytical queries need to be issued over cold data

to extract insightful results (EMC 2014). As a nearline storage device with access latency that is five orders of magnitude larger than HDD, tape will impose a significant performance penalty for even latency-insensitive batch analytic workloads (Kathpal and Yasa 2014). Thus, today, enterprises are faced with a trade-off, as they can store cold data in tape-based archival tier at the expense of performance or in the HDD-based capacity tier and trade-off cost.

C

Key Research Findings

Over the past few years, storage hardware vendors and researchers have become cognizant of the gap between the HDD-based capacity tier and the tape-based archival tier. This has led to the emergence of a new class of nearline storage devices explicitly targeted at cold data workloads. These devices, also referred as *cold storage devices (CSD)*, have two salient properties that distinguish them from the tape-based archival tier. First, they use archival-grade, high-density, shingled magnetic recording-based HDD as the storage media instead of tapes. Second, they explicitly trade off performance for power consumption by organizing hundreds of disks in a massive array of idle disk (MAID) configuration that keeps only a fraction of HDD powered up at any given time (Colarelli and Grunwald 2002).

CSD differ dramatically with respect to price, performance, and peak power consumption characteristics. Some CSD enforce a strict upper limit on the number of HDD that can be spun up at any given point to limit peak power draw. By doing so, they right-provision hardware resources, like in-rack cooling and power management, to cater to the subset of disks that are spun up, reducing operational expenses further. For instance, Pelican (Balakrishnan et al. 2014) packs 1,152 SMR disks in a 52U rack for a total capacity of 5 PB. However, only 8% of disks are spun up at any given time due to restrictions enforced by in-rack cooling and power budget. Similarly, each OpenVault Knox (Yan 2013) CSD stores 30 SMR HDDs in a 2U chassis of which only one can be spun up to minimize the sensitivity

of disks to vibration. Other CSD, like Spectra ArcticBlue Spectra Logic (2013), provide more performance flexibility at the expense of cost, similar to traditional MAID arrays, by only spinning down idle disks while permitting a much larger subset of disks to be active simultaneously.

The net effect of these limitations is that the latency to access data stored in the CSD depends on whether the data resides in a disk that is spun up or down. Access to data in any of the spun-up disks can be done with latency and bandwidth comparable to that of the traditional capacity tier. For instance, Pelican, OpenVault Knox, and ArcticBlue are all capable of saturating a 10 Gb Ethernet link as they provide between 1 and 2 GB/s of throughput with an access latency of 5–10 milliseconds for reading data from spun-up disks. However, accessing data on a spun-down disk takes 5–10 s, as the target disk needs to be spun up before data can be accessed.

CSD form a perfect middle ground between HDD and tape. Due to the use of high-density disks and MAID techniques, CSD are touted to offer cost/GB comparable to tape. With worst-case access latencies in seconds, CSD are closer to HDD than tape with respect to performance. However, CSD are not a drop-in replacement for HDD (Borovica-Gajic et al. 2016). In order to effectively exploit CSD for reducing the cost of cold data storage and analytics, cold storage infrastructures need to design three components of both CSD and analytic engines to work in concert with the shared goal of minimizing the number of disk spin-ups, namely, the CSD storage manager, the CSD I/O scheduler, and the analytic engine's query executor.

CSD Storage Manager

The CSD storage manager is responsible for implementing the storage layout that maps data to disks. Designing an optimal data layout for CSD is a complex problem, as one has to balance performance, reliability, and availability simultaneously. From performance point of view, data that is accessed together should be stored on a set of disks that can spin up together so that requests for such data can be serviced concurrently without any group switches. From the reliability

aspect, the disk chosen for storing data should span multiple failure domains, like different trays in the rack, so that a failure in any single domain does not result in total data loss. With respect to availability, data layout should ensure that recovery from a disk failure is seamless and quick, as disk failures will be a norm rather than an exception in CSD.

In CSD like Pelican (Balakrishnan et al. 2014), where the power and cooling infrastructure of the CSD enforces a strict limit on which set of disks can be spun up at any given time, the choice of data layout is a direct consequence of these restrictions. Pelican (Balakrishnan et al. 2014) assembles disks into groups such that disks within each group belong to different failure domains and can be spun up simultaneously. Each data object stored by Pelican is striped across disks within a single group, using erasure coding to protect data in case of disk failures. By using disks that can be active at the same time, Pelican meets the performance requirement, as multiple object requests can be served concurrently due to striping. Erasure coding helps in improving availability, as recovery after disk failures can be done within a group without any spin-ups and several disks can participate in data restoration.

A storage layout that uses historic data access patterns as a hint for future accesses could further improve performance by packing multiple objects that are always accessed together into a few disk groups. The task of identifying an optimal mapping of objects to disks given a history of accesses can be modeled as an optimization problem (Reddy et al. 2015). However, given the large number of objects stored in CSD, it can be quite complicated to handle each object individually in the optimization algorithm. Thus, the problem of choosing a data layout is decomposed into two subproblems: (i) identifying clusters of objects that are accessed together from the trace and (ii) identifying an optimal mapping of objects to disks. The goal of the first step is to transform individual object access patterns into group access patterns by clustering objects that are accessed together. The optimization algorithm can then map clusters to disks. It has been shown that such an access-driven data layout approach can save

up to 78% power over random placement (Reddy et al. 2015).

CSD I/O Scheduler

The goal of the CSD's I/O scheduler is to service object requests from different clients in a way that minimizes the number of group switches while at the same time ensuring fairness across clients. The CSD group scheduling problem can be reduced to the single-head tape scheduling problem in traditional tertiary storage systems, where it has been shown that an algorithm that picks the tape with the largest number of pending requests as the target to be loaded next performs within 2% of the theoretically optimal algorithm (Prabhakar et al. 2003). If efficiency was the only goal, we could apply such an algorithm, which we henceforth refer to as *Max-Requests*, to the CSD case by picking the disk group with the maximum number of requests. However, the algorithm would not provide fairness, as a continuous stream of requests for a few popular disks can starve out requests for less popular ones.

Pelican CSD solves this problem by scheduling object requests in a *first-come first-serve (FCFS)* order to provide fairness with some parameterized slack that occasionally violates the strict FCFS ordering by reordering and grouping requests to the same disk group to improve performance (Balakrishnan et al. 2014). However, by building on FCFS, Pelican's algorithm trades off efficiency for fairness. The Skipper framework (Borovica-Gajic et al. 2016) introduces a new algorithm that strikes a balance between FCFS and Max-Requests by associating a rank with each group that is computed by considering both the number of requests that need to be serviced for that disk group, and the average time spent by requests waiting for that disk group to spin up. The scheduler always picks the group with the highest rank as the target to be spun up next. By using the number of requests in computing the rank, the scheduler behaves efficiently similar to the Max-Requests algorithm. By including the average waiting time, the scheduler ensures that a request to a less popular group that has been not been serviced for a while will get prioritized over other requests to more popular groups.

Database Query Executor

When a CSD is used as the storage back end for an analytical database engine, query execution will result in the database reading data objects directly from the CSD. In the best case, these database read requests are always serviced from a disk group that is spun up. In such a case, there would be no performance difference between using a CSD and the traditional HDD-based capacity tier. However, in the pathological case, every data access request issued by the database would incur a spin-up delay and cripple performance. Unfortunately, the average case is more likely to be similar to the pathological case due to two assumptions made by traditional databases: (1) storage subsystem has exclusive control over data allocation, and (2) the underlying storage media support random accesses with uniform access latency.

Today, most enterprise databases run in a virtualized setting in a private cloud hosted on premise or in the public cloud. In such a virtualized environment, a CSD will store data corresponding to several databases by virtualizing available storage behind an object interface, similar to OpenStack Swift (Oracle 2015) or Amazon S3 (Amazon 2015). As each database reads objects from the CSD, it has no control over the CSD data layout which is performed by the CSD storage manager. The lack of control over data layout implies that the latency to access a set of relations depends on the way they are laid out across disk groups.

Moreover, the CSD services requests from multiple databases simultaneously. Thus, even if all data corresponding to a single database is located in a single group, the execution time of a single query is not guaranteed to be free of disk spin-ups. This is because the access latency of any database request depends on the currently loaded group, which depends on the set of requests from other databases being serviced at any given time. Thus, in a virtualized enterprise data center that uses CSD as a shared service, both assumptions made by analytical database engines are invalidated leading to suboptimal query execution (Borovica-Gajic et al. 2016).

Exploiting the cost benefits of CSD while minimizing the associated performance trade-off requires eliminating unnecessary disk spin-ups. The only way of achieving such an optimal data access pattern is to have the database issue requests for all necessary data upfront so that the CSD can return data back in an order that minimizes the number of disk spin-ups. Thus, the order in which database receives data, and hence the order in which query execution happens, should be determined by the CSD to minimize the performance impact of group switches.

Unfortunately, current databases are not designed to work with such a CSD-driven query execution approach. Traditionally, databases have used a strict optimize-then-execute model to evaluate queries. The database query optimizer uses cost models and statistics gathered to determine the optimal query plan. Once the query plan has been generated, the execution engine then invokes various relational operators strictly based on the generated query plan with no run-time decision-making. This results in pull-based query execution where the database explicitly requests data in an order determined by the query plan. This pull-based execution approach is incompatible with the CSD-driven approach, as the optimal order chosen by the CSD for minimizing group switches is different from the ordering specified by the query optimizer.

In order to enable CSD-driven query execution, analytical database engines should adopt push-based query execution instead of a pull-based one. There are two approaches that can be used to perform push-based query execution. The first approach is to tightly couple query execution with I/O scheduling in such a way that the scheduler knows and delivers only objects that are required for the query executor to make progress. For instance, in a two-table hash join, the scheduler would deliver objects corresponding to the build relation over which the hash table is built before delivering the probe relation. Such an approach was used by tertiary database engines (Sarawagi and Stonebraker 1996) and makes two assumptions: (i) the database has exclusive access to the storage device, and (ii) the I/O scheduler on the storage device is aware

of database query execution. Unfortunately, both assumptions are not true today with CSD, as CSD storage is exposed behind an object interface and shared among multiple databases.

The second approach, exemplified by the Skipper framework (Borovica-Gajic et al. 2016), is to perform CSD-driven, push-based, out-of-order execution of queries by building on work done in *adaptive query processing (AQP)* (Deshpande et al. 2007). *Multiway join (MJoin)* (Viglas et al. 2003) is one such AQP technique that enables out-of-order execution by using an n-ary join and symmetric hashing to probe tuples as data arrives, instead of using blocking binary joins whose ordering is predetermined by the query optimizer. However, the incremental nature of symmetric hashing in traditional MJoin requires buffering all input data, as tuples that are yet to arrive could join with any of the already received tuples. Thus, in the worst-case scenario of a query that involves all relations in the dataset, the MJoin cache must be large enough to hold the entire dataset. This requirement makes traditional MJoin inappropriate for the CSD use case as having a buffer cache as large as the entire dataset defeats the purpose of storing data on the CSD. Skipper solves this problem by redesigning both MJoin and database buffer caching algorithms to work in concert such that MJoin can perform query execution even with limited cache capacity (Borovica-Gajic et al. 2016).

Examples of Application

With the right combination of layout, I/O scheduling, and query execution, cold data analytic engines like Skipper have demonstrated that it is possible to mask the long access latency of CSD for latency-insensitive batch analytic workloads. Thus, the obvious application for CSD is as the medium of choice for storing cold data, and a natural way of integrating CSD in the four-tier storage hierarchy shown in Fig. 1 is to use it for building a new cold storage tier that stores only cold data.

However, CSD could potentially have a much larger impact on the storage hierarchy based

on two observations. First, with latency-critical workloads running in the performance tier, the HDD-based capacity tier is already limited to batch analytic workloads today. Second, due to the use of high-density SMR disks and MAID techniques, CSD can offer cost and capacity comparable to the tape-based archival tier. Given these two observations, it might be feasible to get rid of the capacity and archival tiers by using a single consolidated cold storage tier. Such an approach would result in the four-tier hierarchy shown in Fig. 1 being reduced to a three-tier hierarchy with DRAM or SSD in the performance tier, CSD in the cold storage tier, and tape in the backup tier. Recent studies report that enterprises can lower their total cost of ownership (TCO) between 40% and 60% by using such storage consolidation (Borovica-Gajic et al. 2016). In terms of absolute savings, these values translate to hundreds of thousands of dollars for a 100TB database and tens of millions of dollars for larger PB-sized databases.

The implications and benefits of using CSD are also applicable to cloud service providers. For instance, cloud providers have already started deploying custom-built CSD for storing cold data (Bandaru and Patiejunas 2015; Bhat 2016; Google 2017). An interesting area of future work involves exploring implementation and pricing aspects of cloud-hosted “cheap-analytics-over-CSD-as-a-service” solutions. Such services would benefit both customers and providers alike, as providers can increase revenue by expanding their storage-as-a-service offering to include cheap analytic services and customers can reduce the TCO by running latency-insensitive analytic workloads on data stored in CSD.

Future Directions for Research

Systems like Pelican (Balakrishnan et al. 2014) and Skipper (Borovica-Gajic et al. 2016) are a first essential step toward broader adoption of cold data analytics, in general, and CSD in particular. There are several avenues of future work with respect to both cold storage hardware and data analytic software.

Over the past few years, several other systems have been built using alternate storage media to reduce the cost of storing cold data. For instance, DTStore (Lee et al. 2016) uses LTFS tape archive for reducing the TCO of online multimedia streaming services by classifying data based on access pattern and storing cold data in tape drives. ROS (Yan et al. 2017) is a rack-scale optical disk library that provides PB-sized storage with in-line accessibility for cold data using thousands of optical disks packed in a single 42U rack. Nakshatra (Kathpal and Yasa 2014) enables Hadoop-based batch analytic workloads to run directly over data stored in tape archives. Similar to Skipper, Nakshatra also modifies both the tape archive’s I/O scheduler and the MapReduce run time to perform push-based execution to hide the long access latency of tape drives.

Today, it is unclear as to how these alternative storage options fare with respect to SMR-HDD-based CSD as the storage media of choice for storing cold data. Some storage media, like optical disks, have the shortcoming that their sequential access bandwidth is substantially lower than HDD or tape. Tape and optical media also have very high access latency compared to CSD. However, an interesting artifact of using push-based engines like Skipper is that query execution becomes much less sensitive to the long access latency of storage devices. Similarly, optical disks are traditionally known to have longer life span compared to HDD and tape. However, it is possible to deal with even frequent media failures by trading off some storage capacity for improved reliability by using erasure coding and data replication techniques in software. Thus, further research is required to understand the implications of storing cold data on these storage devices by running realistic cold data analytic workloads and performing a comparative study of performance and reliability.

On the software side, current systems like Skipper and Nakshatra only target a simplified subset of SQL or MapReduce-based analytic workloads. They do not support complex business intelligence workloads that require operators like data cube (Gray et al. 1997), data mining workloads like clustering and

classification, or iterative machine learning workloads. They also do not consider the availability of additional layers of persistent storage that could be used as caches or prefetch buffers for data stored in CSD or as an extended memory that can be used to swap out intermediate data structures generated by operators like MJoin. Further research is also required to understand how push-based execution can be extended to these workloads.

References

- Amazon (2015) Amazon simple storage service. <https://aws.amazon.com/s3/>. Accessed 1 Oct 2017
- Appuswamy R, Borovica-Gajic R, Graefe G, Ailamaki A (2017) The five-minute rule thirty years later, and its impact on the storage hierarchy. In: Proceedings of the eighth international workshop on accelerating analytics and data management systems using modern processor and storage architectures, Munich
- Balakrishnan S, Black R, Donnelly A, England P, Glass A, Harper D, Legtchenko S, Ogus A, Peterson E, Rowstron A (2014) Pelican: a building block for exascale cold data storage. In: Proceedings of the 11th USENIX conference on operating systems design and implementation, Berkeley, pp 351–365
- Bandaru K, Patiejunas K (2015) Under the hood: Facebook's cold storage system. Facebook. <https://code.facebook.com/posts/1433093613662262/-under-the-hood-facebook-s-cold-storage-system-/>. Accessed 1 Oct 2017
- Bhat S (2016) Introducing azure cool blob storage. Microsoft. <https://azure.microsoft.com/en-us/blog/introducing-azure-cool-storage/>. Accessed 1 Oct 2017
- Borovica-Gajic R, Appuswamy R, Ailamaki A (2016) Cheap data analytics using cold storage devices. Proc VLDB Endow 9(12):1029–1040. <https://doi.org/10.14778/2994509.2994521>
- Colarelli D, Grunwald D (2002) Massive arrays of idle disks for storage archives. In: Proceedings of the ACM/IEEE conference on supercomputing, Los Alamitos, pp 1–11
- Deshpande A, Ives Z, Raman V (2007) Adaptive query processing. J Found Trends Databases 1(1):1–140. <https://doi.org/10.1561/1900000001>
- EMC (2014) The digital universe of opportunities: rich data and the increasing value of the internet of things. <https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf>. Accessed 1 Oct 2017
- Fontana R, Decad G (2015) Roadmaps and technology reality. Presented at the library of congress storage meetings on designing storage architectures for digital collections, Washington, 9–10 Sept 2015
- Google (2017) Nearline cloud storage. <https://cloud.google.com/storage/archival/>. Cited 1 Oct 2017
- Gray J, Graefe G (1997) The five-minute rule ten years later, and other computer storage rules of thumb. SIGMOD Rec 26(4):63–68. <https://doi.org/10.1145/271074.271094>
- Gray J, Graefe G (2007) The five-minute rule twenty years later and how flash memory changes the rules. In: Proceedings of the 3rd international workshop on data management on new hardware, New York, pp 1–9
- Gray J, Putzolu GR (1987) The 5-minute rule for trading memory for disk accesses and the 10-byte rule for trading memory for CPU time. SIGMOD Rec 16(3): 395–398. <https://doi.org/10.1145/38714.38755>
- Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. J Data Min Knowl Discov 1(1):29–53. <https://doi.org/10.1023/A:1009726021843>
- Kathpal A, Yasa GAN (2014) Nakshatra: towards running batch analytics on an archive. In: Proceedings of 22nd IEEE international symposium on the modelling, analysis and simulation of computer and telecommunication systems, Paris, pp 479–482
- Lee J, Ahn J, Park C, Kim J (2016) DTStorage: dynamic tape-based storage for cost-effective and highly-available streaming service. In: Proceedings of the 16th IEEE/ACM international symposium on cluster, cloud and grid computing, Cartagena, pp 376–387
- LTO Ultrium (2015) LTO roadmap. <http://www.ltoultrium.com/lto-ultrium-roadmap/>. Accessed 1 Oct 2017
- Mendoza A (2013) Cold storage in the cloud: trends, challenges, and solutions. <https://www.intel.com/content/www/us/en/storage/cold-storage-atom-xeon-paper.html>. Accessed 1 Oct 2017
- Moore F (2015) Tiered storage takes center stage. Horison Inc. <http://horison.com/publications/tiered-storage-takes-center-stage>. Accessed 1 Oct 2017
- Moore F (2016) Storage outlook 2016. Horison Inc. <https://horison.com/publications/storage-outlook-2016>. Accessed 1 Oct 2017
- Nandkarni A (2014) IDC worldwide cold storage taxonomy. <http://www.idc.com/getdoc.jsp?containerId=246732>. Accessed 1 Oct 2017
- Oracle (2015) OpenStack swift interface for oracle hierarchical storage manager. <http://www.oracle.com/us/products/servers-storage/storage/storage-software/solution-brief-sam-swift-2321869.pdf>. Accessed 1 Oct 2017
- Prabhakar S, Agrawal D, Abbadi A (2003) Optimal scheduling algorithms for tertiary storage. J Distrib Parallel Databases 14(3):255–282. <https://doi.org/10.1023/A:1025589332623>
- Reddy R, Kathpal A, Basak J, Katz R (2015) Data layout for power efficient archival storage systems. In: Proceedings of the workshop on power-aware computing and systems, Monterey, pp 16–20
- Robert Y, Vivien F (2009) Introduction to scheduling, 1st edn. CRC Press, Boca Raton

- Sarawagi S, Stonebraker M (1996) Reordering query execution in tertiary memory databases. In: Proceedings of the 22th international conference on very large data bases, San Francisco, pp 156–167
- Spectra Logic (2013) Spectra arcticblue overview. <https://www.spectralogic.com/products/arcticblue/>. Accessed 1 Oct 2017
- Viglas SD, Naughton JF, Burger F (2003) Maximizing the output rate of multi-way join queries over streaming information sources. In: Proceedings of the 29th international conference on very large data bases, Berlin, pp 285–296
- Yan M (2013) Cold storage hardware v0.5. http://www.opencompute.org/wp/wp-content/uploads/2013/01/Open_Compute_Project_Cold_Storage_Specification_v0.5.pdf. Accessed 1 Oct 2017
- Yan W, Yao J, Cao Q, Xie C, Jiang H (2017) ROS: a rack-based optical storage system with inline accessibility for long-term data preservation. In: Proceedings of the 12th European conference on computer systems, Belgrade, pp 161–174

Clojure

Nicolas Modrzyk
Karabiner Software, Tokyo, Japan

It's easy to get sidetracked with technology, and that is the danger, but ultimately you have to see what works with the music and what doesn't. In a lot of cases, less is more. In most cases, less is more.

Herbie Hancock

Definitions

How to use some best features of Clojure, a LISP like functional programming language, to target big data analysis.

Introduction

Rich Hickey's Clojure language is not in the new section of hyped languages anymore, which is great news for everybody interested in Clojure. The language has just reached [version 1.9](#), and the amount of strong and precise development that went into this new version deserved an ova-

tional praise. But why would you use Clojure nowadays? What can it do for you that the available trillions of other languages available on the market cannot do, or not as efficiently, with programming in Java 1.9, Python, or NodeJS getting all the hype and the tooling nowadays, with even big consulting firms entering the NodeJS market. Clojure being a LISP has a fantastic minimal core and a very small set of functions and data structures. Of course, you have asynchronous thread-safe parallel queues, of course you have parallel processing of all the data structures, and it is also a breeze to go for test-driven development, but what's more is that Clojure changes the way you think about programming and write programs in general. It makes you think in small, versatile, reusable building block. Eventually the users of Clojure, the people writing code with it, do not really think about it anymore; it just becomes the way they naturally code, whatever the language. Clojure also makes your code smaller, easier to debug, maintain, and push to production with less fear of being woken up at 3 AM for issues. Your program is consistent and reliable. Let's review some of the Clojure features and ecosystem that makes Clojure a language of choice for handling big data.

Clojure for Instant Prototyping, with a REPL

Many languages have a REPL nowadays, a Read-Eval-Print-Loop, like a shell prompt, where you can write and execute code block by block. Clojure is not the only language armed with a REPL, but it is one of the most convenient because many functions consist of a simple code block that usually holds on one line.

Take the Fibonacci sequence, for example, it does hold as a one liner

```
(def fib-seq
  ((fn rfib [a b]
    (lazy-seq
      (cons a
        (rfib b (+ a b))))))
  0 1))
```

and is still easily readable.

You usually start a REPL with Leiningen (or boot), the two most famous build tools that actually install Clojure at the same time and make you in command of a build environment at the same time.

Starting a REPL with:

```
lein repl
```

or

```
boot repl
```

And you can copy paste the code above and retrieve the first n elements of the Fibonacci sequence, with the result printed below.

```
(take 10 fib-seq)
;(0 1 1 2 3 5 8 13 21 34)
```

When working with Clojure, programmers would usually always have a REPL running to try ideas on the run or in a business meeting.

Clojure for Code as Data

Beyond having a REPL, Clojure being a LISP has its syntax and its code execution commands virtually identical. Clojure code is made of successive lists of lists of symbols.

What does that mean?

At the REPL, since you have it open, if you want to perform the simple Math of adding 1 to 1, you would write: (remember the LISP prefix notation ...)

```
(+ 1 1)
; that returns 2
```

The code is made of a single sequence, surrounded by parenthesis, and the sequence itself is made of 3 characters, +, 1, and 1.

What the REPL does is converting the string you have written into a sequence and then evaluating its content. To perform the same thing the REPL is doing, you can use the function read-string, where read-string takes a string and converts it to a sequence of symbols, similar to an abstract syntax tree.

```
(read-string "(+ 1 1)")
; (+ 1 1)
```

To evaluate the list, you call eval on the list, which executes the code and returns 2 again.

```
(eval
  (read-string "(+ 1 1")))
; 2
```

To run eval on a sequence directly, people usually use the ‘ (quote) symbol, where the quote means this is already a set of symbols as a sequence, no need to read it.

```
(eval '(+ 1 1))
; 2
```

Why is this nice? It makes it really easy to tweak the compilation step found in coding. To achieve that, Clojure provides macros, which are evaluated during the pre-processing phase, transforming the lists of lists of sequence before they are executed. See the plus-one macro below, which adds one to the variable x.

```
(defmacro plus-one [x]
  '(+ 1 ~x))
```

When performing the compiling step of eval, this macro will rewrite the list as a list of 3 symbols, as seen above, and as expanded below:

```
(clojure pprint/print
  (macroexpand '(plus-one 1)))
; (clojure.core/+ 1 1)
```

This is different than writing a simple function, plus-one-fn:

```
(defn plus-one-fn [x] (+ 1 x))
```

In the sense, that the block code is written, and evaluated as is, not rewritten then evaluated. Which takes us to ...

Clojure Threading Macros

Clojure Macros have the ability to rewrite the sequence of lists of lists before they are executed. Two of the most used macros are `->` and `->>`. The first macro, `->`, takes a list of forms and rewrite in a different order. The new order is such that the previous step block comes as the first parameter, and so the second element of the list in the next step. With the simple `(+ 1 1)` example, again this gives:

```
(-> 1
    (+ 1))
; 2
```

The rewritten code can be expanded and seen using Clojure's macroexpand:

```
(macroexpand `(-> 1 (+ 1)))
; (clojure.core/+ 1 1)
```

Those two macros come very handy when handling data. Let's take a new example with `->>`, the macro that rewrites code with the previous code block becoming the last element of next steps list. Say you want to count the sum of all the integers from 1 to 100000, with the `->>` macro, this would be written:

```
(->> 100000
      (range)
      (apply +))
```

or as a one liner:

```
(->> 100000 (range) (apply +))
```

Again, using macroexpand, you would notice this expands to:

```
(apply + (range 100000))
```

But was much more readable. An example of a more complex operation, where:

- a sequence from 0 up to 100000 is created,
- then each element of the sequence is incremented,
- then each element of the sequence is incremented,
- and finally create a vector of the result

The above processing could be written like this:

```
(into []
  (filter
    even?
    (map inc (range 100000))))
```

Or the definitely more readable:

```
(->>
  100000
  (range)
  (map inc)
  (filter even?)
  (into []))
```

Those Clojure threading macros tend to effectively be usable in many places. For example, in the Origami library, which is a Clojure wrapper around OpenCV, transformations can be applied to images in a sequence also using the `->` macro, as shown below:

```
(->
  (imread "doc/cat_in_bowl.jpeg")
  (cvt-color! COLOR_RGB2GRAY)
  (canny! 300.0 100.0 3 true)
  (bitwise-not!)
  (u/resize-by 0.5)
  (imwrite "doc/canny-cat.jpg"))
```

The result of the transformation shown in the pictures below (Figs. 1 and 2).

Clojure Laziness and Reducers

Usually Clojure tries to do the least amount of work as possible. Meaning, lists of lists are lazily evaluated, effectively evaluated only when really required. Laziness is a core feature of Clojure and



Clojure, Fig. 1 Cat in a bowl



Clojure, Fig. 2 Cat in a bowl

of many programmers. In the example below, a binding named lazy-var is created.

```
(def lazy-var (plus-one-fn 3))
```

The binding was created, but it was not really used yet. It will only be used when, for example, it is required to be printed.

```
(println lazy-var);  
4
```

Laziness is used in many functions of Clojure core, so those functions keep a good balance between efficiency and laziness. (Isn't that what we are all trying to achieve anyway?) Reducers, or reducing functions, on the contrary know they will be combined to be applied on collections that will finally turn into a realized result, realized taken in the sense of not-lazy.

And so, knowing this, reducers can efficiently work in parallel across all the elements of collections they are being applied to. Here is the simple threading example again, this time using reducers.

```
(require  
'[clojure.core.reducers :as r])  
  
(->>  
100000  
(range)  
(r/map inc)  
(r/filter even?)  
(into []))
```

It looks pretty much the same on paper, apart from the r/ prefixes added to the map and filter functions. What does not show on paper is the

fact that the r/map and r/filter functions have been running in parallel and, on sizeable collections, the time required to evaluate all the forms is substantially faster.

The [claypool library](#), while not an official Clojure library, is quite convenient to specify the size of the different thread pools used to do parallel processing on collections.

In Clojure, there is an infamous function named pmap that has been known to make great use of all the cores available to the runtime. (And take down clusters by using all the possible resources available.)

In Clojure, to create a sequence of five elements from 0 to 4 and increase all the elements sequentially, map is usually used.

```
(->>  
5  
(range)  
(map inc))
```

To perform the same processing using as many cores as possible, its parallel version named pmap is usually used.

```
(->>  
5  
(range)  
(pmap inc))
```

And to perform the same parallel work again, but using only four cores, the claypoole library version of pmap could be used, this time specifying the thread pool size.

```
(require  
'[com.climate.claypoole :as cp])  
  
(->>  
5  
(range)  
(cp/pmap 4 inc))
```

Clojure for Multi-threading

Reducers and the different versions of pmap are perfect to apply computations in parallel. To perform asynchronous message handling between different blocks of code, Clojure comes with a core library named core.async, available on [github](#).

To present this, a small example made of two asynchronous blocks of code, exchanging messages through a channel will be used.

One asynchronous block will send messages to the channel, while the other will read messages from that same channel and print them out.

This very first example creates a thread and prints message asynchronously by itself.

```
(require
  '[clojure.core.async
    :refer :all])

(thread
  (dotimes [n 10]
    (Thread/sleep 100)
    (println "hello:" n)))
```

If executed at the REPL, as soon as the block of code is evaluated, evaluation does not block and returns a reference to the executing async thread.

In the meantime, the thread goes on with his happy life and starts printing hello messages every 100 ms.

This second example creates two threads. A thread will update an atom, a variable ready for asynchronous updates, by safely increasing its value by one. A second thread safely reads the value of the atom, while it's being updated by the first thread.

```
(def my-value (atom 0))

(thread
  (dotimes [n 10]
    (Thread/sleep 1000)
    (swap! my-value inc)))

(thread
  (dotimes [_ 5]
    (Thread/sleep 1000)
    (println
      "Counter is now:"
      @my-value)))
```

The number of threads can be increased, and in action, no deadlock occurs. Updates to the atom are done using the STM transaction model that prevents deadlock to occur using transaction to perform updates on atoms.

The third asynchronous example removes the use of the intermediate atom, by directly acting on messages sent via a channel.

The first thread still creates messages, but this time sends them to the newly created hi-chan channel, while the second thread reads the value coming through the channel and prints them out.

```
(def hi-chan (chan))

(thread
  (dotimes [n 10]
    (Thread/sleep 1000)
    (>!! hi-chan
      (str "hello:" n)))

(thread
  (dotimes [_ 10]
    (Thread/sleep 1000)
    (println
      "Channel value:"
      (<!! hi-chan))))
```

Here again, both thread code blocks return immediately without blocking the execution of the main thread, making all this feasible to execute at the REPL.

Core.async can also run in the browser via ClojureScript and is a very compelling strategy to create a back buffer for onscreen user events and other asynchronous data requests.

Clojure for the Backend

Many frameworks exist for server and api development in Clojure, but most of them seem to be inspired or paying close attention to the efforts that were put into the ring and compojure mini-frameworks.

Ring itself is an abstraction over HTTP, while Compojure is a routing library that can, and mostly does, run on top of ring.

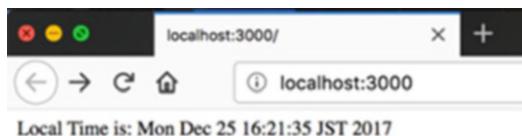
If you have Leiningen installed, getting started with Compojure is as easy as using the compojure project template.

```
lein new compojure intro
```

This creates a prod-ready but also development-ready environment to start writing code.

From the newly created intro folder, a ring development server can be started using the following command.

```
lein ring server
```



Clojure, Fig. 3 Ring HTTP get

This will set up everything needed, apart from your favorite text editor. The main core.clj file in the created project looks deceptively simple, with a single route defined for a GET request on the root context, anything else returning a simple Not Found string.

```
(defroutes app-routes
  (GET "/" [] "Hello World")
  (route/not-found "Not Found"))

(def app
  (wrap-defaults
    app-routes
    site-defaults))
```

The main route can be updated by adding time via a Java date object, using basic Clojure/Java Interop.

```
(defroutes app-routes
  (GET "/" []
    (str
      "Server Time is: "
      (java.util.Date())))
  (route/not-found "Not Found"))
```

This is of course nowhere near a production-ready app, but the point here is that this creates a fully ready development environment in a few minutes, thus bringing server-side prototyping setup down to pretty much zero (Fig. 3).

To get further into the backend programming in Clojure, there is a nice introduction in the blog post below, when you want to quickly grasp all the ring/compojure concepts:

[sinatra-docs-in-clojure](#)

Clojure for the Front End

ClojureScript is Clojure ported to javascript runtime, and why and how to use it is extensively covered in the Reactive with ClojureScript apress book.

The sheer happiness that results from using ClojureScript for the frontend is that this makes your full application code to be written all in Clojure:

- the backend,
- the frontend, and
- the data exchanged between both the backend and the frontend can also be in the Extensible Data Notation, a subset of Clojure.

Clojure's Reagent is a library that acts as a thin wrapper around React, the frontend framework created by Facebook. It removes the extra javascript boiler-plate to make the code very compact, easier to read, write, and maintain.

Reagent can be combined with, among many others, core.async so to get asynchronous looking code (remember only JavaScript thread in the browser) without sacrificing readability.

As for Clojure on the backend, here again there is a Leiningen project template for the reagent framework that can be created with the command below.

```
lein new reagent reagent-intro
```

To get started with the reagent setup, this Leiningen needs two commands to be started, one for the backend and one for the frontend.

Here is the command to start the server development environment:

```
lein ring server
```

Here is the command to start the client development environment:

```
lein figwheel
```

The setup in place, it is now possible to start editing the core.cljs file in the created project.

```
(defn some-component []
  [:div
   [:h3 "I am a component!"]
   [:p.someclass
    "I have " [:strong "bold"]
    [:span {:style {:color "#cc77cc"}}
     " and red"] " text ."]])
```



```
(defn mount-root []
  (reagent/render
    [some-component]))
```

```
(.getElementById js /document
  "app"))

(defn init []
  (mount-root))
```

The code above renders to HTML and is returned to the client as shown in the figure below (Fig. 4).

`init!` is defined and called only once in development mode; thus the `mount-root`, which can be rewritten, is kept separate.

The function `mount-root` itself takes a reagent component and a mount point in the current HTML dom, here `< div id = app >< /div >`, and puts the component at that place.

A component is defined as a standard Clojure function, whose return value is a special kind of markup specific to Clojure, named hiccup.

Hiccup is eventually taken and transformed from a list of Clojure vectors to valid HTML.

Here the component `some-component` output some basic HTML elements like `div`, `h3`, `p`, and `span` elements.

Components can be combined together, so it is possible to create a component from two smaller components, thus being completely in line with Clojure big philosophy of building blocks.

The code below creates a simple Reagent component `top-component`, made of two components of type `some-component`.

```
(defn top-component []
  [:div
   [some-component]
   [some-component]])
```

Note here that even though `some-component` is created using a single common definition, the two mounted `some-component` are different instances.

I am a component!

I have bold and red text.

Also, if not obvious, it is not required to have Clojure on the backend to be able to use ClojureScript and Reagent.

Many now in production projects did a rewrite of their frontend from JavaScript to ClojureScript first, to improve maintainability, without updating the server-side code.

Clojure for Big Data

Sparkling and Flambo are two among the Clojure offers to handle spark data and spark clusters.

Flambo is slightly easier and more importantly makes use of the previously introduced threading macros, `->` and `->>`.

A simple bang-in example of using Flambo is shown below. The example counts the total number of characters of each line, line by line, and then reduces to a single value of the overall total over all the lines.

```
(->
 (f/text-file sc "data.txt")
 ;; returns an unrealized
 ;; lazy dataset
 (f/map (f/fn [s] (count s)))
 ;; returns RDD array
 ;; of length of lines
 (f/reduce (f/fn [x y] (+ x y))))
 ;; use an inline anonymous
 ;; function for reduction
```

Here, `sc` is a Spark Context, which can target a local in-memory cluster, a stand-alone spark cluster, or a mesos cluster.

```
(def c
 (-> (conf/spark-conf)
       (conf/master "local")
       (conf/app-name "infinity")))
 (def sc (f/spark-context c))
```

To perform a spark job, a spark resilient distributed dataset is used, and in the first example presented, the dataset was created by reading from a text file from the local file system.

Creating a Spark RDD can also be done in a few other different ways, like a RDD made from blobs of files stored in HDFS.

```
(def data
 (f/text-file
   sc
   (str
```

```
"hdfs://hostname:port"
"/home/data"
"/*/*.bz2")
```

Of course, using regular Java Interop, it is also possible to target other RDD sources like Cassandra, HBase, and any other storage supported by hadoop.

Flambo itself allows to create valid Spark job that can be deployed and run as any other jobs, but one main limitation it has is that it does not do allow to write and run new job at the REPL without using AOT.

AOT is ahead-of-time compilation meaning you need to statically compile all your code before running, which does prevent it from being created and run at the REPL.

This limitation sparked, pun intended, a new project worth considering named [powderkeg](#), whose goal is to overcome that very limitation.

Clojure for Machine Learning

To go beyond simple big data, and as a recent and strong contender to the TensorFlow framework, Cortex has recently joined the Clojure ecosystem for machine learning.

Some starters and favorites examples to look at to fully understand how Cortex works are: [The cats and dogs sorting example](#) or [the fruits classification example](#).

In this short entry, we will present briefly training a network for xor operations, as it has recently been included in the cortex examples section and is very simple to go over.

As known, xor operations take two values in input and one value in output. This is represented here by creating a dataset to train the network, made of a vector of elements, where each element is a map with an input, :x, and an output, :y, value.

```
(def xor-dataset
  [{:x [0.0 0.0] :y [0.0]}
   {:x [0.0 1.0] :y [1.0]}
   {:x [1.0 0.0] :y [1.0]}
   {:x [1.0 1.0] :y [0.0]}])
```

The neural network to be created will be a linear network of three layers, where the activation

layer will be the hyperbolic tangent activation function. (See: [tanh function](#))

In the definition of the network, input and output layer each specify how many variables are available, and so below:

- 2 for the input, and
- 1 for the output

The network is then created almost by translating written text to a network definition using the cortex api.

```
(def nn
  (network/linear-network
    [(layers/input 2 1 1 :id :x)
     (layers/linear->tanh 10)
     (layers/linear 1 :id :y)]))
```

Now along with the dataset, let's create a trained version of this neural network.

In machine learning, training a network is usually done using two data sets.

- a training set, to tell the network what is expected and correct
- a testing set, to test the network and check whether how it performs.

The training stage is then achieved by passing those two data sets, but here the same dataset is used.

The number of rounds of training is also specified, here 3000 to get an efficient network.

```
(def trained
  (train/train-n
    nn
    xor-dataset
    xor-dataset
    :batch-size 4
    :epoch-count 3000
    :simple-loss-print? false))
```

Each training step outputs some output of the current state of the trained network, notably whether it performed better or not than its own version in the previous step.

```
Training network:
...
Loss for epoch 1:
  (current) 0.7927149 (best)
Saving network to
  trained-network.nippy
```

Finally, to validate results yet another dataset is created, of usually new values, and of course only input values.

```
(def new-dataset
  [{:x [0.0 0.0] }
   {:x [0.0 1.0] }
   {:x [1.0 0.0] }
   {:x [1.0 1.0] }])
```

Then follows running the trained network with this new dataset.

```
(clojure pprint/pprint
  (execute/run
    trained
    new-dataset))
```

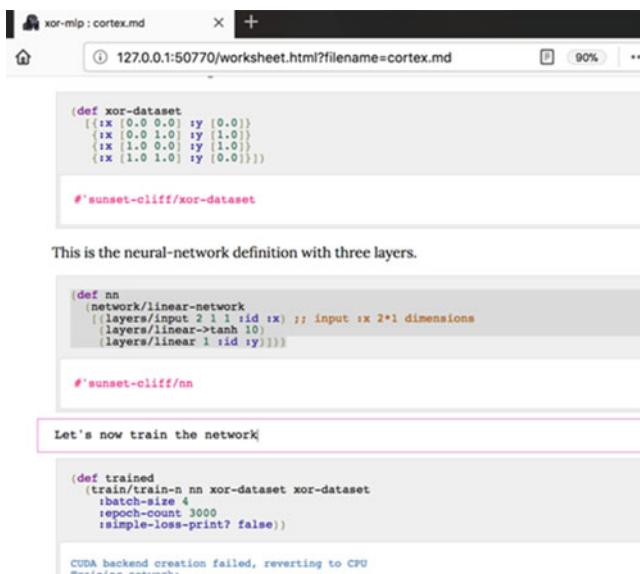
The result is, not surprisingly, very accurate, as shown from a screenshot of the network executed at the REPL.

```
[{:y [0.4397728145122528]}
{:y [0.3689133822917938]}
{:y [0.37379032373428345]}
{:y [0.4282912313938141]}]
```

Clojure for Teaching

Another small in size but very important contribution to the Clojure ecosystem is named Gorilla.

Clojure, Fig. 5 Gorilla



Simply said, Gorilla is a web-based REPL, which allows to create annotated NoteBooks, along ready to be executed code blocks, all in the browser.

It is made of a server-side part that runs a background repl and a simple compojure-based API ready to receive Clojure code as string.

The second part is a Clojure frontend, running a websocket sending Clojure code to the REPL on the server, and waits for the result of the execution and displays it.

The gorilla framework can be added to any of your project using a Leiningen plugin.

This plugin is added either to the project definition project.clj or the global leiningen profile, profile.clj.

```
[lein-gorilla "0.4.0"]
```

Starting the gorilla environment can now be done using a Leiningen command:

```
lein gorilla
```

Gorilla now makes the library and source code available through a web interface. All the presented Cortex code could be run in the browser, for example, and a possible figure is shown below (Fig. 5).

Beyond

Other key Clojure-based projects that can also be looked into are:

- Riemann is an event aggregator, monitoring system entirely written in Clojure. It can receive and process events from almost anything you can imagine.
- Apache Storm almost entirely written in Clojure, reportedly clocks millions of tuples processed per second per node. This is a framework of choice, and if you have complete decision over the technologies used in your next project, comes as a no brainer.
- Datomic, which is a distributed persistence and query model over an historical set of stored facts.

Conclusion

This short entry presented diverse possibilities on how to put Clojure straight into action to:

- Help developers in writing code using a REPL, thus reducing usual life cycle overhead associated with many programming languages
- Reduce maintenance associated with writing code in other languages, by considerably reducing the lines of code used to work and transform data structure by writing macros and/or using the already available threading macros
- Enable pinpoint custom usage of the cores available to each runtime node of your application. Parallel computations is available to most data structures
- core.async allows message passing between different components of your application. Point of interaction can be defined with atom, or channels, and by being reliably highlighted, reduces the coupling between different parts of the code, making the overall end product resistant to changes.
- Front-to-End Clojure brings commonness and portability of the application code thus helping

avoid usual pitfalls when doing client-server programming, in simple sync-ed HTTP requests or real-time websockets.

- Also, not only the code of the application but the projects metadata used by the build tool Leiningen itself was a DSL written in Clojure.
- Bringing entry points to directly interact with any Spark cluster via the REPL, thus making it easier to debug common and more advanced distributed data problems.
- It is also ready for machine learning, providing a real and powerful alternative to TensorFlow with Cortex.
- Finally, the ability to consume all the above, and present annotated pages of work, ready to be distributed and ready for reuse via Gorilla Notebook-style environment.

Cloud Big Data Benchmarks

► [Virtualized Big Data Benchmarks](#)

Cloud Computing for Big Data Analysis

Fabrizio Marozzo and Loris Belcastro
DIMES, University of Calabria, Rende, Italy

Definitions

Cloud computing is a model that enables convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance 2011).

Overview

In the last decade, the ability to produce and gather data has increased exponentially. For ex-

ample, huge amounts of digital data are generated by and collected from several sources, such as sensors, web applications, and services. Moreover, thanks to the growth of social networks (e.g., Facebook, Twitter, Pinterest, Instagram, Foursquare, etc.) and the widespread diffusion of mobile phones, every day millions of people share information about their interests and activities. The amount of data generated, the speed at which it is produced, and its heterogeneity in terms of format represent a challenge to the current storage, process, and analysis capabilities. Those data volumes, commonly referred as Big Data, can be exploited to extract useful information and to produce helpful knowledge for science, industry, public services, and in general humankind.

To extract value from such data, novel technologies and architectures have been developed by data scientists for capturing and analyzing complex and/or high velocity data. In general, the process of knowledge discovery from Big Data is not so easy, mainly due to data characteristics, such as size, complexity, and variety, that are required to address several issues. To overcome these problems and get valuable information and knowledge in shorter time, high-performance and scalable computing systems are used in combination with data and knowledge discovery techniques.

In this context, Cloud computing has emerged as an effective platform to face the challenge of extracting knowledge from Big Data repositories in limited time, as well as to provide an effective and efficient data analysis environment for researchers and companies. From a client perspective, the Cloud is an abstraction for remote, infinitely scalable provisioning of computation and storage resources (Talia et al. 2015). From an implementation point of view, Cloud systems are based on large sets of computing resources, located somewhere “in the Cloud,” which are allocated to applications on demand (Barga et al. 2011). Thus, Cloud computing can be defined as a distributed computing paradigm in which all the resources, dynamically scalable and often virtualized, are provided as services over the Internet. As defined by NIST (National Institute of Stan-

dards and Technology) (Mell and Grance 2011), Cloud computing can be described as: “*A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*” From the NIST definition, we can identify five essential characteristics of Cloud computing systems, which are (i) on-demand self-service, (ii) broad network access, (iii) resource pooling, (iv) rapid elasticity, and (v) measured service.

Cloud computing vendors provide their services according to three main distribution models:

- *Software as a Service (SaaS)*, in which software and data are provided through Internet to customers as ready-to-use services. Specifically, software and associated data are hosted by providers, and customers access them without the need to use any additional hardware or software.
- *Platform as a Service (PaaS)*, in an environment including databases, application servers, development environment for building, testing, and running custom applications. Developers can just focus on deploying of applications since Cloud providers are in charge of maintenance and optimization of the environment and underlying infrastructure.
- *Infrastructure as a Service (IaaS)*, that is an outsourcing model under which customers rent resources like CPUs, disks, or more complex resources like virtualized servers or operating systems to support their operations. Compared to the PaaS approach, the IaaS model has a higher system administration costs for the user; on the other hand, IaaS allows a full customization of the execution environment.

Key Research Findings

Most available data analysis solutions today are based on open-source frameworks, such as

Hadoop (<https://hadoop.apache.org/>) and Spark (<https://spark.apache.org/>), but there are also some proprietary solutions proposed by big companies (e.g., IBM, Kognitio).

Concerning the distribution models of Cloud services, the most common ones for providing Big Data analysis solutions are PaaS and SaaS. Usually, IaaS is not used for high-level data analysis applications but mainly to handle the storage and computing needs of data analysis processes. In fact, IaaS is the more expensive distribution model, because it requires a greater investment of IT resources. On the contrary, PaaS is widely used for Big Data analysis, because it provides data analysts with tools, programming suites, environments, and libraries ready to be built, deployed, and run on the Cloud platform. With the PaaS model, users do not need to care about configuring and scaling the infrastructure (e.g., a distributed and scalable Hadoop system), because the Cloud vendor will do that for them. Finally, the SaaS model is used to offer complete Big Data analysis applications to end users, so that they can execute analysis on large and/or complex datasets by exploiting Cloud scalability in storing and processing data.

As outlined in Li et al. (2010), users can access Cloud computing services using different client devices, Web browsers, and desktop/mobile applications. The business software and user data are executed and stored on servers hosted in Cloud data centers, which provide storage and computing resources. Such resources include thousands of servers and storage devices connected to each other through an intra-Cloud network.

Several technologies and standards are used by the different components of the architecture. For example, users can interact with Cloud services through SOAP-based or RESTful Web services (Richardson and Ruby 2008) and Ajax technologies, which let Cloud services to have look and interactivity equivalent to those provided by desktop applications.

Developing Cloud-based Big Data analysis applications may be a complex task, with specific issues that go beyond those of stand-alone

application programming. For instance, Cloud programming must deal with deployment, scalability, and monitoring aspects that are not easy to handle without the use of ad hoc environments (Talia et al. 2015). In fact, to simplify the development of Cloud applications, specific development environments are often used. Some of the most representative Cloud computing development environments currently in use can be classified into four types:

- *Integrated development environments*, which are used to code, debug, deploy, and monitor Cloud applications that are executed on a Cloud infrastructure, such as Eclipse, Visual Studio, and IntelliJ.
- *Parallel-processing development environments*, which are used to define parallel applications for processing large amount of data that are run on a cluster of virtual machines provided by a Cloud infrastructure (e.g., Hadoop and Spark).
- *Workflow development environments*, which are used to define workflow-based applications that are executed on a Cloud infrastructure, such as Swift and DMCF.
- *Data-analytics development environments*, which are used to define data analysis applications through machine learning and data mining tools provided by a Cloud infrastructure. Some examples are Azure ML and BigML.

The programming model is a key factor to be considered for exploiting the powerful features of Cloud computing. MapReduce (Dean and Ghemawat 2004) is widely recognized as one of the most important programming models for Cloud computing environments, being it supported by Google and other leading Cloud providers such as Amazon, with its Elastic MapReduce service, and Microsoft, with its HDInsight, or on top of private Cloud infrastructures such as OpenNebula, with its Sahara service. Hadoop is the best-known MapReduce implementation, and it is commonly used to develop parallel applications that analyze big amounts of data on Clouds. In fact, Hadoop

ecosystem is undoubtedly one of the most complete solutions for data analysis problem, but at the same time, it is thought for high-skilled users.

On the other hand, many other solutions are designed for low-skilled users or for low-medium organizations that do not want to spend resources in developing and maintaining enterprise data analysis solutions. Two representative examples of such data analysis solutions are Microsoft Azure Machine Learning and Data Mining Cloud Framework.

Microsoft Azure Machine Learning (Azure ML) (<https://azure.microsoft.com/services/machine-learning-studio/>) is a SaaS for the creation of machine learning workflows. It provides a very high level of abstraction, because a programmer can easily design and execute data analytics applications by using simple drag-and-drop web interface and exploiting many built-in tools for data manipulation and machine learning algorithms.

The Data Mining Cloud Framework (DMCF) (Marozzo et al. 2015) is a software system developed at University of Calabria for allowing users to design and execute data analysis workflows on Clouds. DMCF supports a large variety of data analysis processes, including single-task applications, parameter-sweeping applications, and workflow-based applications. A workflow in DMCF can be developed using a visual- or a script-based language. The visual language, called VL4Cloud (Marozzo et al. 2016), is based on a design approach for end users having a limited knowledge of programming paradigms. The script-based language, called JS4Cloud (Marozzo et al. 2015), provides a flexible programming paradigm for skilled users who prefer to code their workflows through scripts.

Other solutions have been created mainly for scientific research purposes, and, for this reason, they are poorly used for developing business applications (e.g., E-Science Central, COMPSs, and Sector/Sphere).

e-Science Central (e-SC) (Hiden et al. 2013) is a Cloud-based system that allows scientists to store, analyze, and share data in the Cloud. It pro-

vides a user interface that allows programming visual workflows in any Web browser.

e-SC is commonly used to provide a data analysis back end to stand-alone desktop or Web applications. To this end, the e-SC API provides a set of workflow control methods and data structures. In the current implementation, all the workflow services within a single invocation of a workflow execute on the same Cloud node.

COMPSs (Lordan et al. 2014) is a programming model and an execution runtime, whose main objective is to ease the development of workflows for distributed environments, including private and public Clouds. With COMPSs, users create a sequential application and specify which methods of the application code will be executed remotely. Providing an annotated interface where these methods are declared with some metadata about them and their parameters does this selection. The runtime intercepts any call to a selected method creating a representative task and finding the data dependencies with all the previous ones that must be considered along the application run.

Sector/Sphere (Gu and Grossman 2009) is an open-source Cloud framework designed to implement data analysis applications involving large, geographically distributed datasets. The framework includes its own storage and compute services, called Sector and Sphere, respectively, which allow to manage large dataset with high performance and reliability.

Examples of Application

Cloud computing has been used in many scientific fields, such as astronomy, meteorology, social computing, and bioinformatics, which are greatly based on scientific analysis on large volume of data. In many cases, developing and configuring Cloud-based applications requires a high level of expertise, which is a common bottleneck in the adoption of such applications by scientists.

Many solutions for Big Data analysis on Clouds have been proposed in bioinformatics, such as Myrna (Langmead et al. 2010), which

is a Cloud system that exploits MapReduce for calculating differential gene expression in large RNA-seq datasets.

Wang et al. (2015) propose a heterogeneous Cloud framework exploiting MapReduce and multiple hardware execution engines on FPGA to accelerate the genome sequencing applications.

Cloud computing has been also used for executing complex Big Data mining applications. Some examples are as follows: Agapito et al. (2013) perform an association rule analysis between genome variations and clinical conditions of a large group of patients; Altomare et al. (2017) propose a Cloud-based methodology to analyze data of vehicles in a wide urban scenario for discovering patterns and rules from trajectory; Kang et al. (2012) present a library for scalable graph mining in the Cloud that allows to find patterns and anomalies in massive, real-world graphs; and Belcastro et al. (2016) propose a model for predicting flight delay according to weather conditions.

Several other works exploited Cloud computing for conducting data analysis on large amount of data gathered from social networks. Some examples are as follows:

You et al. (2014) propose a social sensing data analysis framework in Clouds for smarter cities, especially to support smart mobility applications (e.g., finding crowded areas where more transportation resources need to be allocated); Belcastro et al. (2017) present a Java library, called ParSoDA (Parallel Social Data Analytics), which can be used for developing social data analysis applications.

Future Directions for Research

Some of most important research trends and issues to be addressed in Big Data analysis and Cloud systems for managing and mining large-scale data repositories are:

- *Data-intensive computing.* The design of data-intensive computing platforms is a very significant research challenge with the goal of building computers composed of a large num-

ber of multi-core processors. From a software point of view, these new computing platforms open big issues and challenges for software tools and runtime systems that must be able to manage a high degree of parallelism and data locality. In addition, to provide efficient methods for storing, accessing, and communicating data, intelligent techniques for data analysis and scalable software architectures enabling the scalable extraction of useful information and knowledge from data are needed.

- *Massive social network analysis.* The effective analysis of social network data on a large scale requires new software tools for real-time data extraction and mining, using Cloud services and high-performance computing approaches (Martin et al. 2016). Social data streaming analysis tools represent very useful technologies to understand collective behaviors from social media data. New approaches to data exploration and model visualization are necessary taking into account the size of data and the complexity of the knowledge extracted.
- *Data quality and usability.* Big Data sets are often arranged by gathering data from several heterogeneous and often not well-known sources. This leads to a poor data quality that is a big problem for data analysts. In fact, due to the lack of a common format, inconsistent and useless data can be produced as a result of joining data from heterogeneous sources. Defining some common and widely adopted format would lead to data that are consistent with data from other sources, that means high-quality data.
- *In-memory analysis.* Most of the data analysis tools access data sources on disks, while, differently from those, in-memory analytics access data in main memory (RAM). This approach brings many benefits in terms of query speed up and faster decisions. In-memory databases are, for example, very effective in real-time data analysis, but they require high-performance hardware support and fine-grain parallel algorithms (Tan et al. 2015). New 64-bit operating systems allow to

- address memory up to one terabyte, so making realistic to cache very large amount of data in RAM. This is why this research area has a strategic importance.
- *Scalable software architectures for fine-grain in-memory data access and analysis.* Exascale processors and storage devices must be exploited with fine-grain runtime models. Software solutions for handling many cores and scalable processor-to-processor communications have to be designed to exploit exascale hardware (Mavroidis et al. 2016).
- ## References
- Agapito G, Cannataro M, Guzzi PH, Marozzo F, Talia D, Trunfio P (2013) Cloud4snp: distributed analysis of SNP microarray data on the cloud. In: Proceedings of the ACM conference on bioinformatics, computational biology and biomedical informatics 2013 (ACM BCB 2013). ACM, Washington, DC, p 468. ISBN:978-1-4503-2434-2
- Altomare A, Cesario E, Comito C, Marozzo F, Talia D (2017) Trajectory pattern mining for urban computing in the cloud. *Trans Parallel Distrib Syst* 28(2):586–599. ISSN:1045-9219
- Belcastro L, Marozzo F, Talia D, Trunfio P (2016) Using scalable data mining for predicting flight delays. *ACM Trans Intell Syst Technol.* ACM, New York, 8(1): 5:1–5:20
- Belcastro L, Marozzo F, Talia D, Trunfio P (2016, to appear) Using scalable data mining for predicting flight delays. *ACM Trans Intell Syst Technol (ACM TIST)*
- Belcastro L, Marozzo F, Talia D, Trunfio P (2017) A parallel library for social media analytics. In: The 2017 international conference on high performance computing & simulation (HPCS 2017), Genoa, pp 683–690. ISBN:978-1-5386-3250-5
- Dean J, Ghemawat S (2004) Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design & implementation, OSDI'04, Berkeley, vol 6, pp 10–10
- Gu Y, Grossman RL (2009) Sector and sphere: the design and implementation of a high-performance data cloud. *Philos Trans R Soc Lond A Math Phys Eng Sci* 367(1897):2429–2445
- Hiden H, Woodman S, Watson P, Cala J (2013) Developing cloud applications using the e-science central platform. *Philos Trans R Soc A* 371(1983):20120085
- Kang U, Chau DH, Faloutsos C (2012) Pegasus: mining billion-scale graphs in the cloud. In: 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 5341–5344. <https://doi.org/10.1109/ICASSP.2012.6289127>
- Langmead B, Hansen KD, Leek JT (2010) Cloud-scale rna-sequencing differential expression analysis with Myrna. *Genome Biol* 11(8):R83
- Li A, Yang X, Kandula S, Zhang M (2010) Cloudcmp: comparing public cloud providers. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, pp 1–14
- Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia R (2014) Servicess: an interoperable programming framework for the cloud. *J Grid Comput* 12(1):67–91
- Marozzo F, Talia D, Trunfio P (2015) Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms. *Concurr Comput Pract Exp* 27(17):5214–5237
- Marozzo F, Talia D, Trunfio P (2016) A workflow management system for scalable data mining on clouds. *IEEE Trans Serv Comput*, vol PP(99), p 1
- Martin A, Brito A, Fetzer C (2016) Real-time social network graph analysis using streammine3g. In: Proceedings of the 10th ACM international conference on distributed and event-based systems, DEBS'16. ACM, New York, pp 322–329
- Mavroidis I, Papaefstathiou I, Lavagno L, Nikolopoulos DS, Koch D, Goodacre J, Sourdis I, Papaefstathiou V, Coppola M, Palomino M (2016) Ecoscale: reconfigurable computing and runtime system for future exascale systems. In: 2016 design, automation test in Europe conference exhibition (DATE), pp 696–701
- Mell PM, Grance T (2011) Sp 800-145. The nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Gaithersburg
- Richardson L, Ruby S (2008) RESTful web services. O'Reilly Media, Inc., Newton
- Talia D, Trunfio P, Marozzo F (2015) Data analysis in the cloud. Elsevier. ISBN:978-0-12-802881-0
- Tan KL, Cai Q, Ooi BC, Wong WF, Yao C, Zhang H (2015) In-memory databases: challenges and opportunities from software and hardware perspectives. *SIGMOD Rec* 44(2):35–40
- Wang C, Li X, Chen P, Wang A, Zhou X, Yu H (2015) Heterogeneous cloud framework for big data genome sequencing. *IEEE/ACM Trans Comput Biol Bioinform* 12(1):166–178. <https://doi.org/10.1109/TCBB.2014.2351800>
- You L, Motta G, Sacco D, Ma T (2014) Social data analysis framework in cloud and mobility analyzer for smarter cities. In: 2014 IEEE international conference on service operations and logistics, and informatics (SOLI), pp 96–101

Cloud Databases

► Databases as a Service

Cloud-Based SQL Solutions for Big Data

Marcin Zukowski

Snowflake Computing, San Mateo, CA, USA

Overview

Cloud computing is one of the most important trends in the current software industry. Cloud possesses unique technical and business characteristics, enabling new approaches to software design and new usage models. In this chapter we present the key characteristics of the cloud systems, and discuss opportunities and challenges traditional database systems face when deployed on this new platform. Using a set of existing systems, we demonstrate various approaches to building cloud-based SQL Big Data solutions.

Cloud

Cloud computing is possibly the largest shift in computing since the client-server model became popular. In recent years, we see companies of all sizes embrace it, often for very different reasons. Architecturally, it introduces a lot of previously unavailable features, that provide amazing opportunities to system and application developers. At the same time, careful (re)design of software is needed to take full advantage of them.

The term “Cloud” tends to be used in various contexts:

- An abstract remote location where data is stored, and processing takes place (e.g., “my photos are in the cloud”);
- Infrastructure-as-a-Service (IaaS) solutions, that mostly replace on-premise physical entities like buildings, (virtual) compute and storage resources and networking infrastructure;
- Platform-as-a-Service (PaaS) solutions, extending IaaS with additional services (e.g., database systems, messaging services

etc.) enabling development of end-user applications;

- Software-as-a-Service (SaaS) solutions, encapsulating the complete application functionality in an end-user system;
- Single-vendors cloud systems, e.g., Amazon Web Services (AWS). They typically provide IaaS (e.g., EC2 for AWS) and PaaS (e.g., SQS and Lambda) solutions, and sometimes also end-user SaaS solutions (e.g., AWS Quick-sight). This meaning is used mostly in this article.

Historically, multiple companies made attempts to build cloud platforms. However, the technical complexity and economies of scale led to only a few companies dominating the market (Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform), with a reasonably small number of other players ([Magic Quadrant for Cloud Infrastructure as a Service, Worldwide](#)).

Cloud Platform Characteristics

Cloud systems provide a new, unique set of features not available previously, which offer exciting opportunities for database systems.

Elastic Compute

On-demand provisioning of computing resources is a defining feature of cloud platforms. It allows quickly (typically in minutes) getting access to (effectively) arbitrarily large amount of compute nodes. Additionally, these resources can be as-needed scaled up and down, resulting in users paying only for the actual usage.

This is dramatically different from traditional on-premise situation, where hardware resources:

- Had to be planned months in advance, leading to high capital expenses (CapEx) and slow project rollouts
- Were provisioned for peak usage (further increasing CapEx and reducing cost efficiency)

- or average utilization (leading to insufficient performance at peak times)
- Required continuous maintenance (high operating expense – OpEx)
- Required additional infrastructure (buildings, power) impacting both CapEx and OpEx

Additional elasticity aspects of cloud compute layers include:

- Instance types, e.g., instances with more RAM VS more storage, or with specialized hardware components like GPUs or FPGAs
- Payment models, e.g., fully on-demand or up-front for a given period
- Availability models, e.g., cheaper spot instances in AWS, that can be taken away anytime, and “reserved” instances that provide a better cost efficiency for highly-utilized systems
- Ability to use various storage layers, see below

An elastic compute layer is present in all cloud systems, e.g., AWS EC2, Google Compute Engine and Azure VMs.

Storage Services

Most cloud systems provide multiple layers of storage systems, differing on a number of

dimensions. The table below lists the most popular storage layers from leading providers and their vendor-specific services. These different services can lead to very different software design choices, depending on the requirements of a given system or application (Table 1).

C

Additional Features and Opportunities

Multi-Tenancy A unique aspect of cloud systems is that each cloud platform serves thousands of different users. On the cloud provider side, this leads to economy-of-scale benefits, savings from sharing resources, and higher predictability of general usage trends. On the user side, it gives access to a huge pool of resources. It also opens the possibility of sharing access to the same data, if its stored in a shared layer (like S3). As a result, sharing of data is possible without a physical data movement task (e.g., transferring files) and becomes to a logical operation (e.g., granting access). This allows completely new classes of applications and systems.

Geographical distribution Most cloud systems are internally distributed across multiple (typically 3) physical data centers. As a results many (but not all) services provide resiliency not only to single-machine or single-rack failures, but to the entire data center failures. This provides the

Cloud-Based SQL Solutions for Big Data, Table 1 Example storage services present in various cloud platforms

	Cost	Size	Latency	Bandwidth	Persistency	Access	AWS (Amazon EC2: Storage)	Azure (Introduction to Microsoft Azure Storage)	Google (Google Cloud Platform: Storage Options)
Local instance storage	Free (included)	Limited, fixed	Very low	Very high	Ephemeral	Single instance	Available (not all types)	Available	Available (not all types)
Distributed block storage	Medium	Limited, elastic	Low	High	Persistent	Single instance (at a time)	EBS	General purpose storage	Persistent disks
Distributed file system	High	Unlimited	Medium	Medium	Persistent	Shared	EFS	Azure files	Not available
Object (blob) store	Very low	Unlimited	High	Low	Persistent	Shared	S3	Azure blob storage	Cloud storage

level of availability previously only available to the largest companies.

Security Cloud systems offer improvements to system security at various levels: physical security (state of the art systems deployed at scale); network security (e.g., Virtual Private Cloud in AWS); encryption (automatic storage encryption; encryption key management); authentication and identity management (IAM in AWS); auditing and more. As a result, building secure systems in a cloud is often easier than providing a similar (externally facing) system on-premise.

Additional services Cloud vendors offer dozens of additional managed services, in areas like (examples for AWS): messaging systems (e.g., SMS, SQS), databases (e.g., RDS, DynamoDB), serverless computing (e.g., Elastic Beanstalk, Lambda), management (e.g., CloudTrail, CloudWatch) and more. These powerful tools make creating applications much easier in the cloud environment.

SaaS While IaaS provides flexibility and cost benefits to organizations, a good software-as-a-service product can lead also to a higher productivity, better user satisfaction, and personnel cost reduction. As a result, cloud systems often focus on providing a low-maintenance, mostly automated and easy-to use experience. For example, aspects like friendly user interfaces, automatic software update management, high-availability, built-in monitoring etc are all now expected by most users of cloud systems.

Cloud System Challenges

While providing a lot of great features, cloud system introduce challenges which application designers need to incorporate into their architecture. This section lists some of them.

Unpredictability of single-instance performance Since resources in the cloud are typically shared and virtualized, it is possible for one tenant of the same physical resource to have a negative impact on other users. Modern

virtualization technologies prevent that to a large extent, but the problem is not completely gone.

Increased failure rates similarly, it is more common for instances in the compute layer to die without a warning, due to hardware or software corruption, often caused by activity unrelated to a given user.

Non-transparent system topology For the end-user, the mapping of virtual compute instances onto physical entities is not visible. As a result, one cant say if two nodes might influence each others behavior, or predict the cost of the network communication.

Unpredictability of services stability and performance While cloud services typically offer very high availability, some of them behave inconsistently. For example, it is known that a simple read request to AWS S3, while usually taking a fraction of a second, occasionally can take multiple seconds.

Limited hardware choice Many on premise systems use carefully tuned hardware units, with balanced CPU/RAM/disk and network. While cloud offers multiple instance types, it is often not possible to get the exact hardware configuration that would be optimal for a given application.

Cloud and Database Systems

The previously described properties of the cloud clearly demonstrate these systems are significantly different from on-premise infrastructure available to most companies.

It is perfectly possible to deploy existing software in a cloud environment, and a lot of database systems in fact are deployed this way. However, such strategy often does not allow benefiting from the unique cloud opportunities, and might not handle cloud-specific problems well.

Let us look at a list of various challenges that database system designers face when designing for the cloud

Elasticity Most popular design for on-premise large scale database systems is “shared nothing” (Stonebraker 1985). This is a great architecture for fixed-topology systems, but presents a lot of problems when highly elastic behavior is desired. As a result, other approaches have been proposed.

Scalability Combination of elastic compute resources and pay-for-use billing model, leads to customers demanding higher peak compute power (for a shorter time). As a result, distributed database systems need to be designed to efficiently scale to larger sizes.

Infrastructure reliability In traditional systems, an infrastructure failure (failed disk, network partitioning etc) and performance degradations were often considered exceptional events. In the cloud, they are more likely and systems need to be designed for them.

Performance consistency The performance differences between various nodes participating in query processing can be much higher in a shared environment. As a result, features like load balancing and skew handling are significantly more important in the cloud.

System topology With most of the cloud infrastructure virtualized, the placement of logical entities on physical hardware is often unknown. This causes problems for aspects like fault tolerance, where a single machine failure can potentially influence multiple logical instances, making it much harder to provide system failure guarantees.

Network efficiency Many high-performance distributed database systems are designed for high-performance networking interfaces like InfiniBand – when suddenly faced with e.g., a 1 gigabit Ethernet connection, their performance

can degrade quickly. This influences aspects like distributed algorithms, data exchange formats etc.

Security While being typically deployed in isolated, private environments, databases did not have to worry about many aspects of security present in the cloud environments. For example, for many users, having all data encrypted in-flight is legally required in a shared environment.

Extensibility Many database systems allow using custom software to enhance various features of the system (for example, C or C++ user defined functions). Shared systems need to devise mechanisms protecting against malicious code provided through these mechanisms.

Monitoring On-premise databases could assume the users had system-level access to the instances, giving them insight into aspects like memory or disk utilization. With databases provided as a service, they often do not have this access, and additional levels of monitoring needs to be exposed via separate interfaces.

Client connectivity Databases traditionally assumed the user maintained a network connection throughout their session. However, with web user interfaces and users being more mobile, the need for clients that do not depend on continuous connection arises.

External data Traditional data warehouses could assume they managed all (or most) of the data they needed to access. With the recent data explosion, many organizations keep a lot of data in low-cost services like S3. Ability to efficiently access these becomes highly desired, especially with that data being easily accessible within a given cloud system.

Simplicity Another area where databases do not match the SaaS world perfectly is the system usability. Databases are famous for the complexity of their management, with activities like tuning,

backups, monitoring consume a lot of precious personnel time.

Pricing Traditional database license was reasonably simple, with the user typically paying for the volume of data or the amount of used hardware resources. With the clouds elasticity, new payment models had to be invented, typically focused on actual system usage.

Multi-tenancy Cloud database systems that choose to provide multi-tenant services face additional challenges. For example, workload of one user might adversely influence other users. Additionally, any layers of the system that are shared require strong access control mechanisms. Finally, multi-tenancy can cause scalability challenges for the shared system components.

Example Systems

In this section we discuss a few representative examples of database systems provided as cloud services. While they obviously differ in many database-specific areas, like SQL support, performance etc, we focus on their very different approaches to building a cloud data warehousing system.

Amazon Redshift

Amazon Redshift ([Amazon Redshift](#)) was the first widely available cloud data warehousing system and is still the market leader. It is derived from the on-premise Paraccel database, which followed the traditional shared nothing model (Chen et al. 2009). It uses user-sized EC2 clusters for both processing and storage.

While originally keeping the major design decisions unchanged, Redshift did introduce a number of cloud-focused improvements and extensions (Gupta et al. 2015), mainly:

- Ability to scale up/down and pause/resume, providing much better elasticity than in the on-premise systems

- Integration with S3, simplifying data backups
- Rich user interface for managing and monitoring

Additionally, in 2017 Amazon introduced a major architectural extension for Redshift called Spectrum ([Amazon Redshift Spectrum](#)). It is a sub-system focused on scanning data in S3, where a part of the Redshift query can be pushed to a separate system, which uses highly parallel compute infrastructure for operations like scans, filters and aggregations.

Microsoft Azure SQL Data Warehouse

Azure SQL Data Warehouse (SQLDW) ([Microsoft Azure SQL Data Warehouse](#)) is a cloud-focused evolution of the SQL Server Parallel Data Warehouse ([Parallel Data Warehouse overview](#)), which in turn extended SQL Server with shared-nothing architecture derived from DATAAllegro ([DATAAllegro](#)).

Azure SQLDW uses compute nodes accessing data stored in a distributed block store. All the data is up-front partitioned in the storage layer into a limited (60) set of partitions. Each compute node owns a subset of them, and inter-node data transfer is required to access data owned by other nodes. As such, the data assignment approach and processing layer of Azure SQLDW are similar to shared nothing systems. Still, using a separate storage layer provides some benefits of the shared filesystem approaches. For example redistributing the data between nodes is a logical operations (no data copying is needed) and is relatively fast.

Google BigQuery

Google BigQuery ([Google BigQuery](#)) has a very different lineage than Redshift and Azure SQLDW. It is based on internal Google products ([BigQuery under the hood](#)), mainly Dremel (Melnik et al. 2010) for data processing and Colossus distributed file system ([Fikes](#)) for data storage, following the shared-storage architecture.

Originally Dremel was a reasonably simple internal query processing system with limited functionality (e.g., no real distributed joins;

append-only), with a limited, SQL-like language. After releasing it publicly as BigQuery in 2011, Google has implemented a number of significant improvements bringing it much closer to traditional database systems, including a more-standard SQL support and update capability.

BigQuery is unique in the discussed set of products with its usage model – the user issues a query and only pays for the amount of data that query processes. This is the closest to the pure SaaS approach, with users not having to manage anything related to resource allocation. With this approach, it is also possible for a single query to use tens or even hundreds of machines, depending on what is currently available, with the users cost staying the same.

BigQuery provides a demonstration of how highly distributed compute layer combined with an efficient distributed storage can provide exceptional query performance without any up-front investment.

Snowflake Elastic Data Warehouse

Snowflake ([Introducing Snowflake](#)) is a rare example of a SQL data warehousing system designed from scratch with cloud platforms in mind. Its architecture (Dageville et al. 2016) is driven directly by various attributes of AWS:

- Object store is the cheapest and most reliable storage, leading to using S3 as the persistent data layer
- S3 has no update capability, leading to using immutable micro-partitions as data storage format
- S3 has low performance, leading to columnar storage, heavy data skipping, and local caching
- For compute layer to easily scale up and down, compute nodes should be stateless
- With data in S3, multiple compute nodes can access it at the same time, as long as someone coordinates their activity.
- S3 is not good for frequent data access and compute nodes are stateless, hence an extra management and metadata entity is needed

- In a multi-tenant system, all the barriers are purely logical, hence sharing data between users is possible

The result multi-cluster shared-data architecture uniquely translates a lot of cloud benefits to the end user, providing high elasticity, scalability and novel features.

C

Other Systems

While four systems discussed in this chapter are highly representative examples, there are many other products in this space, providing additional unique cloud-related approaches and features. That list includes, but is not limited to: Teradata Intellicloud ([Teradata IntelliCloud](#)), Micro-Focus Vertica (including the Eon mode) ([Whats New in Vertica 9.0: Eon Mode Beta](#)), Pivotal Greenplum ([Pivotal Greenplum on Amazon Web Services](#)), IBM DashDB ([IBM dashDB](#)), Oracle Autonomous Data Warehouse ([Oracle Autonomous Data Warehouse Cloud](#)) and Amazon Athena ([Amazone Athena](#)).

Conclusions

It seems clear today that most future data processing needs will be fulfilled by cloud systems. This chapter discussed the challenges and opportunities cloud brings, as well as approaches to addressing them. A short overview of a few example systems demonstrates that databases can take very different architectural approaches to this new platform and still build very successful cloud systems. With cloud adoption growing at a rapid pace, we expect a lot of innovation in this space in the coming years.

References

- Amazon EC2: Storage. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html>
Amazon Redshift. <https://aws.amazon.com/redshift>
Amazon Redshift Spectrum. <https://aws.amazon.com/redshift/spectrum/>
Amazone Athena. <https://aws.amazon.com/athena>

BigQuery under the hood. <https://cloud.google.com/blog/big-data/2016/01/bigquery-under-the-hood>

Chen Y et al (2009) Partial join order optimization in the paraccel analytic database In: Proceedings of SIGMOD

Dageville B et al (2016) The snowflake elastic data warehouse In: Proceedings of SIGMOD

DATAAllegro. <https://en.wikipedia.org/wiki/DATAAllegro>

Fikes A Storage architecture and challenges. https://cloud.google.com/files/storage_architecture_and_challenges.pdf

Google BigQuery. <https://cloud.google.com/bigquery>

Google Cloud Platform: Storage Options. <https://cloud.google.com/compute/docs/disks/>

Gupta A et al (2015) Amazon redshift and the case for simpler data warehouses In: Proceedings of SIGMOD

IBM dashDB. <https://www.ibm.com/ms-en/marketplace/cloud-data-warehouse>

Introducing Snowflake. <https://www.snowflake.net/product/>

Introduction to Microsoft Azure Storage. <https://docs.microsoft.com\kern1pt\en-us\kern1pt\azure\kern1pt\storage\kern1pt\common\storage-introduction>

Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. <https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519>

Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T (2010) Dremel: interactive analysis of web-scale datasets. In: Proceedings of VLDB

Microsoft Azure SQL Data Warehouse. <https://azure.microsoft.com/en-us/services/sql-data-warehouse>

Oracle Autonomous Data Warehouse Cloud. https://cloud.oracle.com/en_US/datawarehouse

Parallel Data Warehouse overview. <https://docs.microsoft.com\kern1pt\en-us\kern1pt\sql\kern1pt\analytics\kern1pt-\kern1ptplatform-system\parallel-data-warehouse-overview>

Pivotal Greenplum on Amazon Web Services. <https://pivot.io/partners/aws/pivotal-greenplum>

Stonebraker M (1985) The case for shared nothing. In: Proceedings of HPTS

Teradata IntelliCloud. <https://www.teradata.com/products-and-services/intellicloud>

Whats New in Vertica 9.0: Eon Mode Beta. <https://my.vertica.com/blog/whats-new-vertica-9-0-eon-mode-beta>

Cloudlets

- ▶ [Big Data and Fog Computing](#)

Cluster Scheduling

- ▶ [Advancements in YARN Resource Manager](#)

Clustering of Process Instances

- ▶ [Trace Clustering](#)

CODAIT/Spark-Bench

- ▶ [SparkBench](#)

Collective Schema Matching

- ▶ [Holistic Schema Matching](#)

Columnar Storage Formats

Avrilia Floratou

Microsoft, Sunnyvale, CA, USA

Definitions

Row Storage: A data layout that contiguously stores the values belonging to the columns that make up the entire row.

Columnar Storage: A data layout that contiguously stores values belonging to the same column for multiple rows.

Overview

Fast analytics over Hadoop data has gained significant traction over the last few years, as multiple enterprises are using Hadoop to store data coming from various sources including operational systems, sensors and mobile devices, and web applications. Various Big Data frameworks have been developed to support fast analytics on top of this data and to provide insights in near real time.

A crucial aspect in delivering high performance in such large-scale environments is the underlying data layout. Most Big Data frameworks are designed to operate on top of data stored in various formats, and they are extensible enough to incorporate new data formats. Over the years, a plethora of open-source data formats have been designed to support the needs of various applications. These formats can be *row* or *column* oriented and can support various forms of serialization and compression. The *columnar* data formats are a popular choice for fast analytics workloads. As opposed to row-oriented storage, columnar storage can significantly reduce the amount of data fetched from disk by allowing access to only the columns that are relevant for the particular query or workload. Moreover, columnar storage combined with efficient encoding and compression techniques can drastically reduce the storage requirements without sacrificing query performance.

Column-oriented storage has been successfully incorporated in both disk-based and memory-based relational databases that target OLAP workloads (Vertica 2017). In the context of Big Data frameworks, the first works on columnar storage for data stored in HDFS (Apache Hadoop HDFS 2017) have appeared around 2011 (He et al. 2011; Floratou et al. 2011). Over the years, multiple proposals have been made to satisfy the needs of various applications and to address the increasing data volume and complexity. These discussions resulted in the creation of two popular columnar formats, namely, the Parquet (Apache Parquet 2017) and ORC (Apache ORC 2017) file formats. These formats are both open-source and are currently supported by multiple proprietary and open-source Big Data frameworks. Apart from columnar organization, the formats provide efficient encoding and compression techniques and incorporate various statistics that enable predicate pushdown which can further improve the performance of analytics workloads.

In this article, we first present the major works in disk-based columnar storage in the context of Big Data systems and Hadoop data. We then provide a detailed description of the Parquet and

ORC file formats which are the most widely adopted columnar formats in current Big Data frameworks. We conclude the article by highlighting the similarities and differences of these two formats.

C

Related Work

The first works on columnar storage in the context of Hadoop, namely, the RCFfile (Row Columnar File) (He et al. 2011) and the CIF (Column Input File format) (Floratou et al. 2011) layouts, were mostly targeting the MapReduce framework (Dean and Ghemawat 2008). These two columnar formats have adopted significantly different designs. The CIF file format stored each column in the data as a separate file, whereas the RCFfile adopted a PAX-like (Ailamaki et al. 2001) data layout where the columns are stored next to each other in the same file.

These design choices led to different trade-offs. The CIF file format was able to provide better performance as it allowed accessing only the files that contain the desired columns but required extra logic in order to colocate the files that contain adjacent columns. Without providing colocation in HDFS, reconstructing a record from multiple files would result in high network I/O. The RCFfile, on the other hand, did not require any such logic to be implemented as all the columns were stored in the same HDFS block. However, the RCFfile layout made it difficult to enable efficient skipping of columns due to file system prefetching. As a result, queries that were accessing a small number of columns would pay a performance overhead (Floratou et al. 2011).

The successor of the RCFfile format, namely, the ORC (Optimized Row Columnar) file format, was developed to overcome the limitations of the RCFfile format. It was originally designed to speed up Hadoop and Hive, but it is currently incorporated in many other frameworks as well. The ORC file format still uses a PAX-like (Ailamaki et al. 2001) layout but can more efficiently skip columns by organizing the data in larger blocks called row groups. A detailed

description of the ORC file format is provided in the following section.

The Parquet (Apache Parquet 2017) file format is another popular, open-source columnar format. The Parquet file format was designed to efficiently support queries over deeply nested data. The format is based on the Dremel distributed system (Melnik et al. 2010) that was developed at Google for interactively querying large datasets. Similar to the ORC format, the parquet format has also adopted a PAX-like layout (Ailamaki et al. 2001). The following section presents Parquet’s layout in detail.

Detailed comparisons of various columnar formats can be found in Huai et al. (2013) and in Floratou et al. (2014). The work by Floratou et al. (2014) compares the ORC and Parquet file formats in the context of Hive (Apache Hive 2017) and Impala (Kornacker et al. 2015) for SQL workloads.

Other related open-source technologies are Apache Arrow (2017) and Apache Kudu (2017). Apache Arrow is a columnar in-memory format that can be used on top of various disk-based storage systems and file formats to provide fast, in-memory analytical processing. Note that as opposed to the file formats discussed above, Arrow is an in-memory representation and not a disk-based file format. Apache Kudu is an open-source storage engine that complements HDFS (Apache Hadoop HDFS 2017) and HBase (Apache Hbase 2017). It provides efficient columnar scans as well as inserts and updates. Kudu provides mutable storage, whereas the file formats described above (e.g., Parquet) are immutable, and thus any data modifications require rewriting the dataset in HDFS.

Columnar File Formats for Big Data Systems

In this section, we describe in more detail two popular, open-source columnar formats that have been incorporated in various Big Data frameworks, namely, the **ORC** and the **Parquet** file formats. Both ORC and Parquet are Apache projects (Apache ORC 2017; Apache

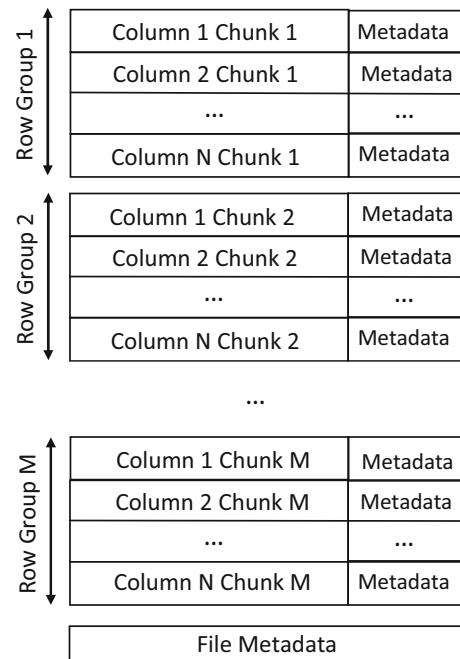
Parquet 2017) and are actively used by multiple organizations when performing analytics over Hadoop data.

The Parquet File Format

The Parquet file format (Apache Parquet 2017) is a self-described columnar data format specifically designed for the Hadoop ecosystem. It is supported by many Big Data frameworks such as Apache Spark (Zaharia et al. 2012) and Apache Impala (Kornacker et al. 2015), among others. The format inherently supports complex nested data types as well as efficient compression and encoding schemes. In the next section, we describe the format layout in more detail.

File Organization

The Parquet format supports both primitive (e.g., integer, Boolean, double, etc.) and complex data types (maps, lists, etc.). The structure of a Parquet file is shown in Fig. 1. The file consists of a set of **row groups** which essentially represent horizontal partitions of the data. A **row group** consists of a set of **column chunks**. Each



Columnar Storage Formats, Fig. 1 The parquet file format

column chunk contains data from a particular data column and it is guaranteed to be contiguous in the file. The column chunk consists of one or more pages which are the unit of compression and encoding. Each page contains a header that describes the compression and encoding method used for the data in the page.

As shown in Fig. 1, the file consists of N columns spread across M row groups. The row groups are typically large (e.g., 1 GB) to allow for large sequential I/Os. Since an entire row group might need to be accessed, the HDFS block size should be large enough to fit the row group. A typical configuration is to have 1 GB HDFS blocks that contain one row group of 1 GB.

Metadata is stored at all the levels in the hierarchy, i.e., file, column chunk, and page. The bottom of the file contains the file metadata which include information about the data schema as well as information about the column chunks in the file. The metadata contains statistics about the data such as the minimum and maximum values in a column chunk or page. Using these statistics, the Parquet file format supports predicate pushdown which allows skipping of pages and reduces the amount of data that needs to be decompressed and parsed. The Parquet readers first fetch the metadata to filter out the column chunks that must be accessed for a particular query and then read each column chunk sequentially.

Compression

As described in the previous section, the unit of compression in the Parquet file format is the page. The format supports common compression codecs such as GZIP and Snappy (Snappy Compression 2017). Moreover, various types of encoding for both simple and nested data types are also supported. The interested reader can find more information about the various encoding techniques in Parquet Encodings (2017).

The ORC File Format

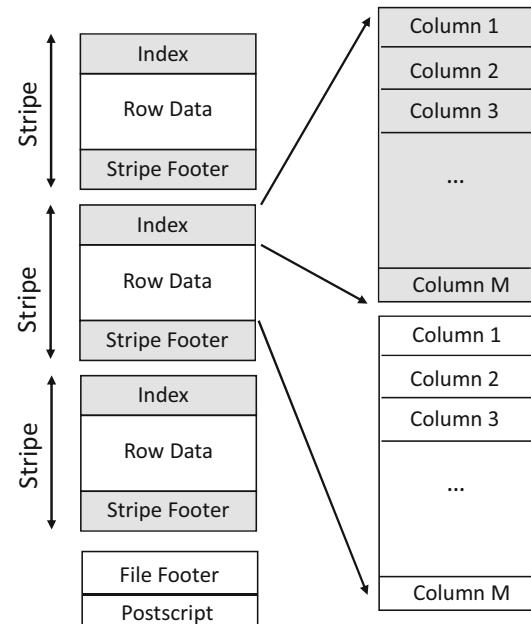
The Optimized Row Columnar (ORC) format (Apache ORC 2017) is a column-oriented storage layout that was created as part of an

initiative to speed up Apache Hive (2017) queries and reduce the storage requirements of data stored in Apache Hadoop (2017). Currently the ORC file format is supported by many Big Data solutions including Apache Hive (2017), Apache Pig (2017), and Apache Spark (Zaharia et al. 2012), among others.

File Organization

The ORC format is an optimized version of the previously used Row Columnar (RC) file format (He et al. 2011). The format is self-describing as it includes the schema and encoding information for all the data in the file. Thus, no external metadata is required in order to interpret the data in the file. The format supports both primitive (e.g., integer, Boolean, etc.) and complex data types (maps, lists, structs, unions). Apart from the columnar organization, the ORC file supports various compression techniques and lightweight indexes.

The structure of an ORC file is shown in Fig. 2. As shown in the figure, the file consists of three major parts: a set of stripes, a file footer, and a postscript. The



Columnar Storage Formats, Fig. 2 The ORC file format

`postscript` part provides all the necessary information to parse the rest of the file such as the compression codec used and the length of the file footer. The `file footer` contains information related to the data stored in the file, such as the number of rows in the file, statistics about the columns, and the data schema. The process of reading the file starts by first reading the tail of the file that consists of the `postscript` and the `file footer` that contain metadata about the main body of the file.

An ORC file stores multiple groups of row data as `stripes`. Each `stripe` has a size of about 250 MB and contains only entire rows so a row cannot span multiple stripes. Internally, each `stripe` is divided into index data, row data, and stripe footer in that order. The data columns are stored next to each other in a PAX-like (Ailamaki et al. 2001) organization. Depending on the query's access pattern, only the necessary columns are decompressed and deserialized. The stripe footer contains the location of the columns in data as well as information about the encoding of each column. The indexes contain statistics about each column in a row group (set of 10,000 rows). For example, in the case of integer columns, the column statistics include the minimum, maximum, and sum values of the column in the given row group. These statistics can be used to skip entire sets of rows that do not satisfy the query predicates. Recently, bloom filters can additionally be used to better prune row groups (ORC Index 2017).

Finally, it is worth noting that column statistics are also stored in the `file footer` at the granularity of `stripes`. These statistics enable skipping entire `stripes` based on a filtering predicate.

Compression

Depending on the user's configuration, an ORC file can be compressed using a generic compression codec (typically zlib (ZLIB Compression 2017) or snappy (Snappy Compression 2017)). The file is compressed in chunks so that entire chunks can be directly skipped without decompressing the data. The default compression chunk size is 256 KB but it is configurable. Having

larger chunks typically results in better compression ratios but requires more memory to be allocated during decompression.

The ORC file makes use of various run-length encoding techniques to further improve compression. The interested reader can find more information about the supported encoding methods in ORC Encodings (2017)

File Format Comparison

The Parquet and ORC file formats have both been created with the goal of providing fast analytics in the Hadoop ecosystem. Both file formats are columnar and they have adopted a PAX-like layout. Instead of storing each column at a separate file, the columns are stored next to each other in the same HDFS block. By having large `row groups` and HDFS blocks, both formats exploit large sequential I/Os and can skip unnecessary columns.

Both formats support various compression codecs at a fine granularity so that the amount of data that is decompressed is minimized. They also support various encoding schemes depending on the data type.

Finally, the file formats incorporate statistics at various levels in order to avoid decompressing and deserializing data that do not satisfy the filtering predicates. Typically, the statistics include the maximum and the minimum values of a column. The ORC file additionally supports bloom filters at the `row group` level, while the Parquet file supports statistics at the page level as well.

Regarding the data types supported, the Parquet file format has been designed to inherently support deeply nested data using the record shredding and assembly algorithm presented in Melnik et al. (2010). The ORC file, on the other hand, flattens the nested data and creates separate columns for each underlying primitive data type.

Conclusions

The increased interest in fast analytics over Hadoop data fueled the development of multiple open-source data file formats. In this work, we

focused on the columnar storage formats that are used to store HDFS data. In particular, we first reviewed the work on disk-based columnar formats for Big Data systems and then presented a detailed description of the open-source ORC and Parquet file formats. These two columnar data formats are currently supported by a plethora of Big Data solutions. Finally, we highlighted the differences and similarities of the two file formats.

Cross-References

- ▶ [Caching for SQL-on-Hadoop](#)
- ▶ [Wildfire: HTAP for Big Data](#)

References

- Ailamaki A, DeWitt DJ, Hill MD, Skounakis M (2001) Weaving relations for cache performance. In: Proceedings of the 27th international conference on very large data bases (VLDB'01), pp 169–180
- Apache Arrow (2017) Apache Arrow. <https://arrow.apache.org/>
- Apache Hadoop (2017) Apache Hadoop. <http://hadoop.apache.org>
- Apache Hadoop HDFS (2017) Apache Hadoop HDFS. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- Apache Hbase (2017) Apache HBase. <https://hbase.apache.org/>
- Apache Hive (2017) Apache Hive. <https://hive.apache.org/>
- Apache Kudu (2017) Apache Kudu. <https://kudu.apache.org/>
- Apache ORC (2017) Apache ORC. <https://orc.apache.org/>
- Apache Parquet (2017) Apache Parquet. <https://parquet.apache.org/>
- Apache Pig (2017) Apache Pig. <https://pig.apache.org/>
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
- Floratou A, Patel JM, Shekita EJ, Tata S (2011) Column-oriented Storage Techniques for MapReduce. Proc VLDB Endow 4(7):419–429
- Floratou A, Minhas UF, Özcan F (2014) SQL-on-Hadoop: full circle back to shared-nothing database architectures. Proc VLDB Endow 7(12):1295–1306
- He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z (2011) RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: Proceedings of the 2011 IEEE 27th international conference on data engineering (ICDE'11). IEEE Computer Society, pp 1199–1208
- Huai Y, Ma S, Lee R, O'Malley O, Zhang X (2013) Understanding insights into the basic structure and essential issues of table placement methods in clusters. Proc VLDB Endow 6(14):1750–1761
- Kornacker M, Behm A, Bittorf V, Bobrovitsky T, Ching C, Choi A, Erickson J, Grund M, Hecht D, Jacobs M, Joshi I, Kuff L, Kumar D, Leblang A, Li N, Pandis I, Robinson H, Rorke D, Rus S, Russell J, Tsirogiannis D, Wanderman-Milne S, Yoder M (2015) Impala: a modern, open-source SQL engine for hadoop. In: CIDR
- Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T (2010) Dremel: interactive analysis of web-scale datasets. Proc VLDB Endow 3(1–2):330–339
- ORC Encodings (2017) ORC Encodings. <https://orc.apache.org/docs/run-length.html>
- ORC Index (2017) ORC Index. <https://orc.apache.org/docs/spec-index.html>
- Parquet Encodings (2017) Parquet Encodings. <https://github.com/apache/parquet-format/blob/master/Encodings.md>
- Snappy Compression (2017) Snappy Compression. [https://en.wikipedia.org/wiki/Snappy_\(compression\)](https://en.wikipedia.org/wiki/Snappy_(compression))
- Vertica (2017) Vertica. <https://www.vertica.com/>
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI, USENIX
- ZLIB Compression (2017) ZLIB Compression. <https://en.wikipedia.org/wiki/Zlib>

Complex Event Processing

- ▶ [Pattern Recognition](#)

Component Benchmark

Klaus-Dieter Lange and David L. Schmidt
Hewlett Packard Enterprise, Houston, TX, USA

Synonyms

[Microbenchmark](#); [Worklet](#)

Overview

Component benchmarks are valuable tools for isolating and analyzing the performance characteristics of specific subsystems within a server

environment. This chapter will provide historical context for component benchmarks, brief descriptions of the most commonly used benchmarks, and a discussion on their uses in the field.

Definitions

Software that utilizes standard applications, or their core routines that focus on a particular access pattern, in order to measure the performance and/or efficiency of one of the primary server and storage components (CPU transactions, memory, and storage/network IO).

Historical Background

From earliest days of digital computer systems, the goal of quantifying, improving, and optimizing computational performance has been a subject great interest. The earliest measures of performance were comparisons of low-level instruction execution times. A mix of several of these execution times would be combined to produce an overall rating that could be compared between systems. The most well-known of these early benchmarks was the Gibson mix, devised by Jack Gibson of IBM (Gibson 1970). As high-level computer languages were developed, more complex applications were created to develop more rigorous methods of measuring a system's performance. Whetstone (Longbottom 2014) and Dhrystone (Weiss 2002) are both early examples of high-level language benchmarks.

As computer systems became more complex, benchmarks were developed to measure the performance of the separate components of the system, such as memory, floating-point arithmetic coprocessors, and data IO. Most of these early benchmarks utilized synthetic workloads and were usually provided to users as source code that needed to be compiled on the system of interest. This allowed such benchmarks to be used on multiple platforms, but there was generally no set standard on how to compile

such benchmarks or how to compare results between different architectures which limited the scope of their use. Industry-standard consortia such as Transaction Processing Performance Council (TPC, <http://www.tpc.org>) and Standard Performance Evaluation Corporation (SPEC, <https://www.spec.org>) were established in the mid-1980s in an effort to provide fair standards for measuring system-level and component-level performance of differing platforms. Several of the benchmarks developed by these consortia are intended to measure component-level performance.

Foundations

Component benchmarks are useful tools to analyze IT equipment and troubleshoot performance bottlenecks. They utilize standard applications, or their core routines that focus on a particular access pattern, in order to measure the performance of one of the primary server and storage components such as CPU transactions, memory access, storage IO, or network IO. By limiting their scope, component benchmarks achieve a deeper analysis of the targeted subsystem. At the same time component benchmarks are often easier to develop than benchmarks that stress the complete server environment and they are typically less expensive to run, because they require fewer equipment and engineering resources.

There can be an overlap in what is considered a component benchmark vs. a microbenchmark. A microbenchmark is typically more limited in scope than a component benchmark, focusing on the performance of a single feature of a component rather than the entire component. Microbenchmarks utilize synthetic workloads rather than actual user applications to measure the performance of their intended resource. Microbenchmarks are further described in a later chapter.

Following is a description of the most common component benchmarks categorized by their target subsystem in a Big Data environment.

Processor subsystem

SPEC CPU2017 (<https://www.spec.org/cpu2017>) is a benchmark package developed by the Standard Performance Evaluation Corporation (SPEC) to provide a comparative measure of compute-intensive performance across the widest practical range of hardware platforms. The benchmark uses workloads developed from programs from a variety of application areas, including artificial intelligence, molecular dynamics, physics, weather-forecasting, imaging, modeling, and computer development. Suites of these workloads are run to measure floating-point (SPECrate2017 Floating Point, SPECspeed2017 Floating Point) and integer (SPECrate2017 Integer, SPECspeed2017 Integer) performance. The suites are further categorized as SPECrate and SPECspeed suites. The SPECrate suites run multiple concurrent copies of each benchmark workload to provide a measure of throughput performance. The SPECspeed suites run a single copy of each benchmark workload to measure processing speed and parallelization performance.

SPEC CPU2006 (<https://www.spec.org/cpu2006>) is the predecessor to SPEC CPU2017 and likewise measures CPU and memory performance utilizing integer and floating-point intensive benchmark suites. There are both rate (SPECint_rate2006, SPECfp_rate2006) and speed (SPECint2006, SPECfp2006) suites to analyze different performance characteristics of the CPU and memory subsystems.

HEP-SPEC06 (Michelotto et al. 2010; HEPiX benchmarking working group) is a benchmark based on the SPEC CPU2006 benchmark suites developed by the HEPiX forum to measure processing performance of applications in a high energy physics environment. HEP-SPEC06 requires the use of the SPEC CPU2006

benchmark tools, but only runs a subset of the component workloads; only the six benchmark modules written in C++ are utilized and are compiled as 32-bit executables. The HEP-SPEC06 run script will then invoke multiple instances of the SPEC CPU2006 run script, each running a single copy of the benchmark workloads. The output results of each SPEC CPU2006 instance are then used to generate a final HEP-SPEC06 score.

Whetstone (Longbottom) was one of the first benchmarks to attempt to standardize the measurement of floating-point performance of CPUs. Developed in 1972, the benchmark workload was a representation of a set of 124 simple Whetstone ALGOL 60 compiler instructions translated into FORTRAN. The benchmark result was measured in thousands of Whetstone instructions per second (kWips), and later in millions of Whetstone instructions per second (MWIPS). The benchmark was updated over the years and ported to C, C++, Basic, and Java. It is frequently included in component benchmark suites.

Dhrystone (Weiss 2002) is a synthetic benchmark program developed in 1984 to measure integer-based processing performance. Developed utilizing metadata from several applications written in different programming languages, Dhrystone was designed to measure system program performance. For many years Dhrystone was considered the representative benchmark for general processor performance. Originally written in Ada, it was translated into C and UNIX and is still often used in component benchmark suites. The name Dhrystone is a pun on the benchmark Whetstone, another benchmark popular at the time of its development.

LINPACK (The LINPACK benchmark programs and reports; Dongarra et al. 2002) is a software library for performing numerical linear algebra on digital com-

puters. The LINPACK benchmark uses the LINPACK software library to solve a set of predetermined set of n by n linear algebra equations in order to measure a system's floating-point computing performance, measured in floating point operations per second (FLOPS). The benchmark workload is almost exclusively floating-point based, so is an excellent stressor of the processors' math and vector instruction sets. There have been several versions of LINPACK benchmarks since their creation in 1979, which include LINPACK 100 ($n = 100$), LINPACK 1000 ($n = 1000$), and HPLinpack (a highly parallelized version that can run across multiple systems). HPL is a portable implementation of the HPLinpack benchmark written in C. Precompiled LINPACK and HPL are available for certain system architectures.

Memory

STREAM (<https://www.cs.virginia.edu/stream>; McCalpin 1995) is a simple synthetic benchmark that measures sustainable memory bandwidth, reported in MB/s. The benchmark is specifically designed to work with dataset much larger than the available processor cache on any given system so that the results are more indicative of the performance of a very large, vector-style application. The benchmark yields four metrics, representing different memory operations: Copy, Add, Scale, and Triad. The benchmark had C and FORTRAN versions available which can run either in a single-threaded or distributed fashion. A precompiled version of this benchmark is often included in component benchmark suites.

Storage IO

SPEC SFS2014 (<https://www.spec.org/sfs2014>) is a benchmark suite which measures file server throughput and response time. The SPEC SFS2014 SP2 benchmark includes five workloads which measure storage performance in different

application areas: database operations, software builds, video data acquisition (VDA), virtual desktop infrastructure (VDI), and electronic design automation (EDA). Each workload is independent of the others and reports the throughput (in MB/s) and overall response time (in msec) for a given number of workload instances. Netmist is the load generator, allowing the benchmark suite can be run over any version of NFS, SMB/CIFS, object-oriented, local, or other POSIX-compatible file system.

Iometer (<http://www.iometer.org>) is a tool which measures the performance and characterization of I/O components for single and clustered systems. It is based on a client-server model where a single client controls one or manager driver which generates I/O using one or more worker threads. Iometer performs asynchronous I/O and can access files or block devices. The tool is very flexible and allows the user to adjust the workload by configuring target disk parameters (e.g., disk size, starting sector, number of outstanding I/Os) and the workload access specifications (e.g., transfer request size, percent random/sequential distribution, percent read/write distribution, aligned I/Os, reply size, TCP/IP status, burstiness). All of the output data is collected into a CSV file for easy manipulation. One of the more commonly utilized output parameter is I/O operations per second (IOPS).

IOzone (http://www.iozone.org/docs/IOzone_msword_98.pdf) is a file system benchmark utility which generates and measures a variety of file operations. These file operations include read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write, pread/pwrite variants, aio_read, aio_write, and mmap. The results can be exported into useful graphs which show performance characteristics and bottlenecks of the disk I/O subsystem. The benchmark is written in C and can be run under many operating systems.

Network IO

Netperf (<https://github.com/HewlettPackard/netperf>) is a software application that provides network bandwidth testing between two hosts on a network. It measures performance of bulk data transfer and request/response network traffic using either TCP or UDP via BSD sockets. There are optional tests that measure performance of DLPI, UnixDomainSockets, the Fore ATM API, and the HP HiPPI LLa interface. Originally developed by Hewlett Packard, it is now available from github. Netperf has been ported to run on numerous distributions of UNIX and Linux, Windows, and VMware.

Component benchmarks are often bundled together into a suite of benchmarks that may or may not also include microbenchmarks to provide a tool that can measure several system features in a single package. Note that these suites are frequently designed for the personal computer or workstation market rather than for the server environment.

Following is a description of the more common component benchmark suites.

SERT (<https://www.spec.org/sert>) is the Server Efficiency Rating Tool developed by SPEC to evaluate the energy efficiency of servers. The SERT suite is a suite of 12 “worklet” modules that stress different system components and measure performance as well as the power consumption of the system. These worklets stress the CPU, memory, and disk I/O subsystem at various target load levels well, as well as one hybrid worklet that stress a series of complex CPU and memory access pattern. The SERT suite utilizes the Chauffeur (<https://www.spec.org/chauffeur-wdk>) harness and is configurable so individual worklets or categories of worklets can be run.

LMBench (<http://lmbench.sourceforge.net>) is a suite of simple benchmarks that measure bandwidth and latency performance for a variety of system components, including memory, caches, disk I/O, and network I/O.

LMBench is written in C and can be used on most distributions of UNIX and Linux.

SiSoftware Sandra (<http://www.sisoftware.eu>) is suite of modules that provide information about a system’s hardware and software, including performance measurements on the CPU, memory, disk I/O, and graphics. Only available on Windows, the primary target of the suite is PC and workstation systems. There are many versions available from the free Lite version to the Enterprise version.

3DMark (<https://www.3dmark.com>) is a benchmarking tool that measures performance of CPU and graphics utilizing modules based off different versions of DirectX. Only available on Windows, Android, and iOS, 3DMark is intended for analyzing PCs used by gaming enthusiasts.

PCMark (<https://www.futuremark.com/benchmarks/pcmark>) is a benchmarking tool similar to 3DMark which measures the performance of different components of PCs, such as CPU, memory, disk I/O, and graphics. Only available on Windows, PCMark is intended for home use.

Key Applications

Single component benchmarks generally are not directly useful for the analysis of Big Data environments, as they measure only one or a few specific aspects of one primary server or storage components. A suite of component benchmarks like the SERT suite can be a key method to measure the behaviors of the primary server and storage components. Nonetheless, they are not measuring the interaction and possible performance bottlenecks between those components. The real usefulness of component benchmarks is their role in the simplification of the performance analysis and the pinpointing of the actual performance bottlenecks within a component.

The major key application of component benchmarks is their usage during server development and the design/deployment phases of new Big Data environments. In general, the first step is to run one of the component suites (or

a series of hand-picked component benchmarks) in order to determine if one subsystem exhibits lower-than-expected performance. Next, an analysis of the subsystem is conducted to further pinpoint the root causes. For example, the performance results from the SERT suite determine lower-than-expected CPU subsystem performance. The next step would include running the SPEC CPU2017 benchmark suite to further close in on the root cause. Finally, after a possible resolution of the issues is implemented, the original component suite should be run again in order to verify that the resolution satisfactorily fixed the bottleneck. Please note that it is the nature of performance bottlenecks that they switch from one component to another, once the root cause of the original bottleneck is resolved.

In order to maintain their usefulness, component benchmarks need a continuous development cycle to keep up with emerging technologies. For example, in order to accurately measure the performance gain of a new microarchitecture, a compiled version of the benchmark code (including its libraries if applicable), which supports this new microarchitecture, should be utilized. A 0.5–3.0% performance gain from such a microarchitecture optimization is quite common.

Cross-References

- ▶ [Analytics Benchmarks](#)
- ▶ [Benchmark Harness](#)
- ▶ [Business Process Performance Measurement](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [Energy Benchmarking](#)
- ▶ [Energy Efficiency in Big Data Analysis](#)
- ▶ [Graph Benchmarking](#)
- ▶ [Machine Learning Benchmarks](#)
- ▶ [Metrics for Big Data Benchmarks](#)
- ▶ [Microbenchmark](#)
- ▶ [Performance Evaluation of Big Data Analysis](#)
- ▶ [SparkBench](#)
- ▶ [Stream Benchmarks](#)
- ▶ [Virtualized Big Data Benchmarks](#)
- ▶ [YCSB](#)

References

- 3DMark benchmark. <https://www.3dmark.com>
- Chauffeur harness. <https://www.spec.org/chauffeur-wdk>
- Dongarra JJ, Luszczek P, Petitet A (2002) The LINPACK benchmark: past, present, and future. http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpl_paper.pdf
- Gibson JC (1970) The Gibson mix. Technical report TR 00.2043. IBM Systems Development Division, Poughkeepsie
- HEPiX benchmarking working group. <https://www.hepix.org/e10227/e10327/e10325>
- Iometer project. <http://www.iometer.org>
- IOzone filesystem benchmark. http://www.iozone.org/docs/IOzone_msword_98.pdf
- LMBench tool. <http://lmbench.sourceforge.net>
- Longbottom (2014) R. Whetstone benchmark history and results. <http://www.roylongbottom.org.uk/whetstone.htm>
- McCalpin JD (1995) Memory bandwidth and machine balance in current high performance computers. IEEE Comput Soc Tech Comm Comput Arch (TCCA) NewsL. https://www.researchgate.net/publication/213876927_Memory_Bandwidth_and_Machine_Balance_in_Current_High_Performance_Computers
- Michelotto M, Alef M, Iribarren A, Meinhard H, Wegner P, Bly M, Benelli G, Brasolin F, Degaudenzi H, De Salvo A, Gable I, Hirstius A, Hristov P (2010) A comparison of HEP code with SPEC benchmark on multi-core worker nodes. <http://www.pd.infn.it/hepmark/HS06.pdf>
- Netperf benchmark. <https://github.com/HewlettPackard/netperf>
- PCMark benchmark. <https://www.futuremark.com/benchmarks/pcmark>
- Server Efficiency Rating Tool (SERT). <https://www.spec.org/sert>
- SiSoftware Sandra. <http://wwwsisoftware.eu>
- SPEC CPU 2006 benchmark. <https://www.spec.org/cpu2006>
- SPEC CPU 2017 benchmark. <https://www.spec.org/cpu2017>
- SPEC SFS 2014 benchmark. <https://www.spec.org/sfs2014>
- Standard Performance Evaluation Corporation (SPEC). <https://www.spec.org>
- STREAM benchmark. <https://www.cs.virginia.edu/stream>
- The LINPACK benchmark programs and reports. <http://www.netlib.org/benchmark/index.html>
- Transaction Processing Performance Council (TPC). <http://www.tpc.org>
- Weiss A (2002) Dhrystone benchmark: history, analysis, scores and recommendations. <http://www.johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf>

Compressed Indexes for Repetitive Textual Datasets

Travis Gagie¹ and Gonzalo Navarro²

¹EIT, Diego Portales University, Santiago, Chile

²Department of Computer Science, University of Chile, Santiago, Chile

Definitions

Given a text or collection of texts containing long repeated substrings, we are asked to build an index that takes space bounded in terms of the size of a well-compressed encoding of the text or texts and that, given an arbitrary pattern, can quickly report the occurrences of that pattern in the dataset.

Overview

Humanity now stores as much data in a year as we did in our whole history until the turn of the millennium. Most applications that use this data need to query it efficiently, and one of the most important kinds of queries is pattern matching in texts. It is usually impractical to scan massive textual datasets every time we want to count or find the occurrences of a pattern, so we must index them. Until fairly recently, indexing a text often took much more memory than simply storing the dataset. In 2000, however, Ferragina and Manzini (2000, 2005) showed how to simultaneously compress and index a text, with the index itself supporting access to the text and thus replacing it. Also in 2000, Grossi and Vitter (2000, 2005) and Sadakane (2000, 2003) proposed an alternative data structure with essentially the same query functionality but with a space bound proportional to the size of the uncompressed text; it was then improved to occupy compressed space. Ferragina and Manzini's and Grossi and Vitter's data structures, the FM-index and compressed suffix array (CSA), have

had a dramatic impact on several fields such as bioinformatics, where FM-indexes are central to many important DNA aligners such as Bowtie (Langmead et al. 2009) and BWA (Li and Durbin 2009). We note that, in contrast to inverted lists, CSAs and FM-indexes can index any kind of text and allow any pattern to be sought, without regard for word boundaries. This is important not only for applications involving DNA but also to index source code repositories, multimedia sequences, and even tokenized word sequences in order to perform phrase searches.

CSAs and FM-indexes use space comparable to the sizes of encodings produced by the best statistical compressors, which achieve bounds in terms of higher-order entropies, but massive texts are often repetitive and are thus much more compressible using dictionary-based compressors such as LZ77 (Ziv and Lempel 1977). For example, humans are genetically almost identical, so a good dictionary-based compressor can compress a database of a thousand human genomes to about 1% of its size, as reported by the 1000 Genomes Project (<http://www.internationalgenome.org>). Naturally, researchers have tried to adapt CSAs and FM-indexes to take better advantage of repetitions or to find alternatives to them that do so. Their efforts can be broadly classified into four groups: run-length compressed FM-indexes, indexes based on Lempel-Ziv compression or context-free grammars, compressed directed acyclic word graphs (CDAWGs), and, most recently, graph-based indexes. We briefly survey the history of each approach, mentioning its strengths and weaknesses and giving pointers to practical implementations where they are available.

Key Research Findings

Run-Length Compressed CSAs and FM-Indexes

Consider a repetitive text (or the concatenation of the texts in a repetitive collection) $T[1..n]$. The suffix array of T , $A[1..n]$, points to the starting

positions of all the suffixes of T in lexicographic order. Assume a long string $S[1..k]$ appears s times in T . Then the s suffixes starting with $S[1..k]$ appear in a contiguous range $A[p_1..p_1 + s - 1]$. It is likely that the suffixes starting with $S[2..k]$ also appear together in another range $A[p_2..p_2 + s - 2]$, in the same order as those in $A[p_1..p_1 + s - 1]$. The same is likely to occur for any $S[k'..k]$ as long as $k - k'$ is sufficiently large. Such a regularity shows up in the Ψ function of CSAs, $\Psi[i] = A^{-1}[A[i] + 1]$, since the values in $\Psi[p_1..p_1 + s - 1]$ will point to $[p_2..p_2 + s - 2]$, and so on, thereby inducing runs of 1s in $\Psi'[i] = \Psi[i] - \Psi[i - 1]$, which is the main component of CSAs. It also shows up in the Burrows-Wheeler Transform (BWT) of T , $\text{BWT}[i] = T[A[i] - 1]$, which is the main component of the FM-index: $\text{BWT}[p_2..p_2 + s - 1]$ will be a run of copies of $S[1]$ and so on.

Mäkinen et al. (2010) took the first step toward adapting CSAs and FM-indexes to handle repetitive texts better, by run-length compressing the runs of 1s in Ψ' or the runs of the same letters in the BWT. The number of runs in Ψ' is almost the same as the number r of runs in the BWT. Their run-length compressed CSA and FM-index then use $O(r)$ space, often much less than a standard CSA or FM-index while still counting patterns' occurrences quickly.

Until very recently it was not known how to compress similarly the suffix-array sample with which we can locate patterns' occurrences in T , without making those locating queries extremely slow. However, Gagie et al. (2018) have now introduced a new sampling scheme that takes $O(r)$ space and lets us locate each occurrence with essentially only a predecessor query. Specifically, they can count the number of occurrences of a pattern of length m in $O(m \log \log n)$ time and then locate each occurrence in $O(\log \log n)$ time.

Lempel-Ziv and Grammar-Based Indexes

Kärkkäinen and Ukkonen (1996) showed how to store an $O(z)$ -space data structure in addition to the text T , where z is the number of phrases in the LZ77 parse of T , such that locating all the occurrences of a pattern of length m takes $O(m^2 + m \log z + \sqrt{mz \log m})$ time plus $O(\log z)$ time per occurrence. They divided the

occurrences into two types, called primary and secondary: the former start in one phrase and end in another (or at the next phrase boundary), and the latter are completely contained within single phrases. Their idea was to store a pair of Patricia trees, one for the reversed phrases in the parse and the other for the suffixes starting at phrase boundaries. By dividing a pattern every possible way into a nonempty prefix and a (possibly empty) suffix and searching for the reversed prefix in the first tree and the suffix in the second, they obtain a pair of Patricia tree nodes or, equivalently, a pair of ranges in the lexicographically sorted reversed prefixes and suffixes. Using an $O(z)$ -space data structure for four-sided two-dimensional range reporting, they obtain all the primary occurrences. They use access to the text to verify the occurrences. Using another $O(z)$ -space data structure that represents the structure of the LZ77 parse, they can then obtain all the secondary occurrences from the primary ones.

Researchers are still following Kärkkäinen and Ukkonen's basic design. The major alterations have aimed at a compressed representation of the text, which is essential in the repetitive scenario. Compressed representations, which must offer direct access in order to support searches, have been built on the LZ77 parse or a variant of it (Kreft and Navarro 2013; Do et al. 2014; Navarro 2017), a context-free grammar (Claude and Navarro 2011, 2012; Gagie et al. 2012, 2014; Bille et al. 2017), a run-length compressed FM-index without samples (which also replaces the Patricia trees) (Belazzougui et al. 2015, 2017), and CDAWGs (Belazzougui et al. 2015, 2017). Another improvement has been to substitute z-fast tries for the Patricia trees, which allows removing the quadratic dependence on m in the query time (Gagie et al. 2014; Bille et al. 2017).

There are also indexes based on edit-sensitive parsing (Maruyama et al. 2013; Takabatake et al. 2014, 2016) and locally consistent parsing (Nishimoto et al. 2016), although the former has poor worst-case query time and seems practical only for long patterns, and the latter is complicated and competitive in theory only when the text is dynamic.

No structure using $O(z)$ space offers good worst-case query times, because it is not known how to support fast access to the text within this space. The most recent theoretical bounds are due to Bille et al. (2017), who showed how we can store an $O(z \log(n/z) \log \log z)$ -space data structure such that later we can locate all the occurrences of a pattern of length m in $O(m)$ time plus $O(\log \log z)$ time per occurrence. The implementation of Kreft and Navarro (2013) uses $O(z)$ space and offers the best compression with query times that are competitive in practice (despite their $O(m^2 z)$ worst-case bound).

Several authors (Schneeberger et al. 2009; Wandelt et al. 2013; Ferrada et al. 2014; Procházka and Holub 2014; Rahn et al. 2014) have independently proposed ideas essentially equivalent to finding the primary occurrences using an FM-index built on the substrings of length 2ℓ centered around phrase boundaries, where ℓ is a parameter. Gagie and Puglisi (2015) surveyed early work in this direction, and others (Danek et al. 2014; Wandelt and Leser 2015; Valenzuela 2016; Valenzuela and Mäkinen 2017; Ferrada et al. 2018) have given new implementations. These indexes are practical, though they have good worst-case query times only for patterns of length $m \leq \ell$. Increasing ℓ at construction worsens compression, whereas searching for patterns longer than ℓ requires the use of heuristics.

CDAWGs

A string $S[1..k]$ appearing s times in T tends to induce large isomorphic subtrees in the suffix tree of T , namely, the subtrees (with s leaves) of the suffix tree nodes representing the strings $S[1..k]$, $S[2..k]$, and so on. By regarding the suffix tree as a deterministic automaton, identifying all of its leaves with a single final state, and minimizing it, one obtains the CDAWG of T (Blumer et al. 1987). The minimization gets rid of the isomorphic subtrees, so the CDAWG benefits from repetitiveness as well.

The CDAWG can be represented using $O(e)$ space, where e is the number of right extensions of maximal repeats in T . With it, we can locate all the occurrences of a pattern of length m in

$O(m \log \log n)$ time plus $O(1)$ time per occurrence (Belazzougui et al. 2015). Its disadvantage is that e is always larger than r and z , by a wide margin in practice, and hence CDAWGs tend to be much larger than the structures described above. The fastest variant (Belazzougui and Cunial 2017) uses $O(e + \bar{e})$ space (where \bar{e} is the e measure of the reversed text) and answers locating queries in optimal time: $O(m)$ and then $O(1)$ per occurrence.

An advantage of CDAWGs is that they can be augmented to support suffix tree functionality (Belazzougui and Cunial 2017; Takagi et al. 2017), which is more powerful than just counting and locating pattern occurrences and of interest in bioinformatics applications. The CDAWG implements a number of suffix tree navigation operations in $O(\log n)$ time. Other compressed suffix trees for repetitive collections (Abeliuk et al. 2013; Navarro and Ordóñez 2016) build on a run-length encoded CSA or FM-index and apply grammar compression to the extra suffix tree components: the suffix tree's topology and the longest common prefix array. Like the CDAWG, they profit from repetitiveness but are significantly larger than their underlying compressed suffix array. The only compressed suffix tree of this kind offering good space guarantees (Gagie et al. 2017b), uses $O(r \log(n/r))$ space, and implements the suffix tree operations in time $O(\log(n/r))$.

Graph-Based Indexes

Graph-based indexes are the newest approach to compressed indexing of repetitive textual datasets so far focused on genomics (Eggertsson et al. 2017; Novak et al. 2017b). Most are generalizations of FM-indexes; Na et al.'s suffix tree of an alignment (Na et al. 2013a) and suffix array of an alignment (Na et al. 2013b) have now been superseded by their FM-index of an alignment (Na et al. 2016; Na et al. 2018). Work in this direction began with Ferragina et al. (2009) extending FM-indexes to labelled trees, followed by Bowe et al. (2012) extending them to de Bruijn graphs and Sirén et al. (2011, 2014) and Sirén (2017) extending them to labelled directed acyclic graphs (DAGs). Gagie

et al. (2017a) recently introduced a framework unifying these results.

Sirén et al.’s indexes are meant for pan-genomics and work essentially by embedding and indexing the DAGs in a kind of de Bruijn graph, allowing matches in recombinations of the input genomes. On the other hand, Na et al. index only the input genomes, essentially by embedding and indexing them in a kind of colored de Bruijn graph.

Both techniques depend on the presence of long, perfectly conserved regions to obtain better compression than run-length compressed FM-indexes, so it is not clear whether they will retain a significant advantage when compressing databases of tens of thousands of genomes in which low-frequency variations are included, how they compare to reference-free alignment methods, and to what extent they have applications outside bioinformatics.

Implementations

The run-length compressed FM-index of Mäkinen et al. (2010) is available at <http://jltsiren.kapsi.fi/rindex>. There are several implementations specifically for genomics (e.g., BGT, BWT-merge, MSBWT, RopeBWT2, SGA), many of which are available on GitHub. The index of Gagie et al. (2018) is implemented at <https://github.com/nicolaprezza/r-index>. The best implementation of the index of Kreft and Navarro (2013) is by Claude et al. (2016), currently available at <https://github.com/migumar2/uiHRDC/tree/master/self-indexes/LZ>.

Valenzuela (2016) and Valenzuela and Mäkinen (2017) give a good general-purpose hybrid index and a good pan-genomic aligner based on a hybrid index, available at <https://www.cs.helsinki.fi/u/dvalenzu/software>.

The DAG-based implementation of Sirén et al. is available at <https://github.com/jltsiren/gcsa2>. There are also several implementations of graph indexes specifically for genomics (Dilthey et al. 2015; Maciuca et al. 2016; Novak et al. 2017a; Paten et al. 2017), some of which are available at <https://github.com/vgteam/vg>.

Example Application

The main applications of indexing repetitive texts have been in bioinformatics, such as indexing genomic databases to support fast alignments. We may be given a database of a thousand human genomes, for example, and asked to preprocess it such that later, given a pattern and an integer k , we can quickly report which genomes contain substrings within edit distance k of that pattern. There are several techniques for implementing such approximate pattern matching using indexes for exact matching (Navarro and Raffinot 2002; Ohlebusch 2013; Mäkinen et al. 2015).

Future Directions for Research

There are several important open problems in the area. An immediate one is to explore the practical impact of the new run-length compressed FM-index of Gagie et al. (2018). In the longer term, combining run-length compressed FM-indexes with graph-based indexes and finding new applications for such indexes seem promising. Another challenge is to upgrade suffix arrays to suffix trees for repetitive collections; suffix trees require access to arbitrary suffix array cells, which is not known to be possible within $O(r)$ space, where r is again the number of runs in the BWT. Obtaining better search performance within space close to $O(z)$ in Lempel-Ziv- and grammar-based indexes is also important, where z is again the number of phrases in the LZ77 parse, because z is in practice significantly smaller than r , even if in theory they are incomparable. String attractors (Kempa and Prezza 2017) were introduced as a generalization of these two and many other measures of repetitiveness, so indexes based on them could be good with respect to all these measures simultaneously.

Cross-References

- ▶ [Genomic Data Compression](#)
- ▶ [Grammar-Based Compression](#)
- ▶ [Inverted Index Compression](#)

References

- Abeliuk A, Cánovas R, Navarro G (2013) Practical compressed suffix trees. *Algorithms* 6(2):319–351
- Belazzougui D, Cunial F (2017) Representing the suffix tree with the CDAWG. In: Proceedings of the 28th symposium on combinatorial pattern matching (CPM), pp 7:1–7:13
- Belazzougui D, Cunial F, Gagie T, Prezza N, Raffinot M (2015) Composite repetition-aware data structures. In: Proceedings of the 26th symposium on combinatorial pattern matching (CPM), pp 26–39
- Belazzougui D, Cunial F, Gagie T, Prezza N, Raffinot M (2017) Flexible indexing of repetitive collections. In: Proceedings of the 13th conference on computability in Europe (CiE), pp 162–174
- Bille P, Ettienne MB, Gørtz IL, Vildhøj HW (2017) Time-space trade-offs for Lempel-Ziv compressed indexing. In: Proceedings of the 28th symposium on combinatorial pattern matching (CPM), pp 16:1–16:17
- Blumer A, Blumer J, Haussler D, McConnell RM, Ehrenfeucht A (1987) Complete inverted files for efficient text retrieval and analysis. *J ACM* 34(3): 578–595
- Bowe A, Onodera T, Sadakane K, Shibuya T (2012) Succinct de Bruijn graphs. In: Proceedings of the 12th workshop on algorithms in bioinformatics (WABI), pp 225–235
- Claude F, Navarro G (2011) Self-indexed grammar-based compression. *Fundamenta Informaticae* 111(3): 313–337
- Claude F, Navarro G (2012) Improved grammar-based compressed indexes. In: Proceedings of the 19th symposium on string processing and information retrieval (SPIRE), pp 180–192
- Claude F, Fariña A, Martínez-Prieto MA, Navarro G (2016) Universal indexes for highly repetitive document collections. *Inf Syst* 61:1–23
- Danek A, Deorowicz S, Grabowski S (2014) Indexes of large genome collections on a PC. *PLoS One* 9(10):e109384
- Dilthey A, Cox C, Iqbal Z, Nelson MR, McVean G (2015) Improved genome inference in the MHC using a population reference graph. *Nat Genet* 47(6): 682–688
- Do HH, Jansson J, Sadakane K, Sung W (2014) Fast relative Lempel-Ziv self-index for similar sequences. *Theor Comput Sci* 532:14–30
- Eggertsson HP et al (2017) Graphyper enables population-scale genotyping using pangenome graphs. *Nat Genet* 49(11):1654–1660
- Ferrada H, Gagie T, Hirvola T, Puglisi SJ (2014) Hybrid indexes for repetitive datasets. *Phil Trans R Soc A* 372(2016):20130137
- Ferrada H, Kempa D, Puglisi SJ (2018) Hybrid indexing revisited. In: Proceedings of the 20th workshop on algorithm engineering and experiments (ALENEX), pp 1–8
- Ferragina P, Manzini G (2000) Opportunistic data structures with applications. In: Proceedings of the 41st symposium on foundations of computer science (FOCS), pp 390–398
- Ferragina P, Manzini G (2005) Indexing compressed text. *J ACM* 52(4):552–581
- Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2009) Compressing and indexing labeled trees, with applications. *J ACM* 57(1):4:1–4:33
- Gagie T, Puglisi SJ (2015) Searching and indexing genomic databases via kernelization. *Front Bioeng Biotechnol* 3:12
- Gagie T, Gawrychowski P, Kärkkäinen J, Nekrich Y, Puglisi SJ (2012) A faster grammar-based self-index. In: Proceedings of the 6th conference on language and automata theory and applications (LATA), pp 240–251
- Gagie T, Gawrychowski P, Kärkkäinen J, Nekrich Y, Puglisi SJ (2014) LZ77-based self-indexing with faster pattern matching. In: Proceedings of the 11th Latin American symposium on theoretical informatics (LATIN), pp 731–742
- Gagie T, Manzini G, Sirén J (2017a) Wheeler graphs: a framework for BWT-based data structures. *Theor Comput Sci* 698:67–78
- Gagie T, Navarro G, Prezza N (2017b) Optimal-time text indexing in BWT-runs bounded space. Technical report 1705.10382, arXiv.org
- Gagie T, Navarro G, Prezza N (2018) Optimal-time text indexing in BWT-runs bounded space. In: Proceedings of the 29th symposium on discrete algorithms (SODA), pp 1459–1477
- Grossi R, Vitter JS (2000) Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In: Proceedings of the 32nd symposium on theory of computing (STOC), pp 397–406
- Grossi R, Vitter JS (2005) Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J Comput* 35(2): 378–407
- Kärkkäinen J, Ukkonen E (1996) Lempel-Ziv parsing and sublinear-size index structures for string matching. In: Proceedings of the 3rd South American workshop on string processing (WSP), pp 141–155
- Kempa D, Prezza N (2017) At the roots of dictionary compression: string attractors. In: Proceedings of the 50th symposium on theory of computing (STOC), 2018. CoRR abs/1710.10964
- Kreft S, Navarro G (2013) On compressing and indexing repetitive sequences. *Theor Comput Sci* 483: 115–133
- Langmead B, Trapnell C, Pop M, Salzberg SL (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* 10(3):R25
- Li H, Durbin R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25(14):1754–1760

- Maciuca S, del Ojo Elias C, McVean G, Iqbal Z (2016) A natural encoding of genetic variation in a Burrows-Wheeler transform to enable mapping and genome inference. In: Proceedings of the 16th workshop on algorithms in bioinformatics (WABI), pp 222–233
- Mäkinen V, Navarro G, Sirén J, Välimäki N (2010) Storage and retrieval of highly repetitive sequence collections. *J Comput Biol* 17(3):281–308
- Mäkinen V, Belazzougui D, Cunial F, Tomescu AI (2015) Genome-scale algorithm design: biological sequence analysis in the era of high-throughput sequencing. Cambridge University Press, Cambridge
- Maruyama S, Nakahara M, Kishiiue N, Sakamoto H (2013) ESP-index: a compressed index based on edit-sensitive parsing. *J Discrete Algorithms* 18:100–112
- Na JC, Park H, Crochemore M, Holub J, Iliopoulos CS, Mouchard L, Park K (2013a) Suffix tree of alignment: an efficient index for similar data. In: Proceedings of the 24th international workshop on combinatorial algorithms (IWOCA), pp 337–348
- Na JC, Park H, Lee S, Hong M, Lecroq T, Mouchard L, Park K (2013b) Suffix array of alignment: a practical index for similar data. In: Proceedings of the 20th symposium on string processing and information retrieval (SPIRE), pp 243–254
- Na JC, Kim H, Park H, Lecroq T, Léonard M, Mouchard L, Park K (2016) FM-index of alignment: a compressed index for similar strings. *Theor Comput Sci* 638:159–170
- Na JC, Kim H, Min S, Park H, Lecroq T, Léonard M, Mouchard L, Park K (2018) FM-index of alignment with gaps. *Theor Comput Sci.* <https://doi.org/10.1016/j.tcs.2017.02.020>
- Navarro G (2017) A self-index on block trees. In: Proceedings of the 17th symposium on string processing and information retrieval (SPIRE), pp 278–289
- Navarro G, Ordóñez A (2016) Faster compressed suffix trees for repetitive text collections. *J Exp Algorithmics* 21(1):article 1.8
- Navarro G, Raffinot M (2002) Flexible pattern matching in strings – practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge, UK
- Nishimoto T, Tomohiro I, Inenaga S, Bannai H, Takeda M (2016) Dynamic index and LZ factorization in compressed space. In: Proceedings of the prague stringology conference (PSC), pp 158–170
- Novak AM, Garrison E, Paten B (2017a) A graph extension of the positional Burrows-Wheeler transform and its applications. *Algorithms Mol Biol* 12(1):18:1–18:12
- Novak AM et al (2017b) Genome graphs. Technical report 101378, bioRxiv
- Ohlebusch E (2013) Bioinformatics algorithms: sequence analysis, genome rearrangements, and phylogenetic reconstruction. Oldenbusch Verlag, Bremen, Germany
- Paten B, Novak AM, Eizenga JM, Garrison E (2017) Genome graphs and the evolution of genome inference. *Genome Res* 27(5):665–676
- Procházka P, Holub J (2014) Compressing similar biological sequences using FM-index. In: Proceedings of the data compression conference (DCC), pp 312–321
- Rahn R, Weese D, Reinert K (2014) Journaled string tree – a scalable data structure for analyzing thousands of similar genomes on your laptop. *Bioinformatics* 30(24):3499–3505
- Sadakane K (2000) Compressed text databases with efficient query algorithms based on the compressed suffix array. In: Proceedings of the 11th international symposium on algorithms and computations (ISAAC), pp 410–421
- Sadakane K (2003) New text indexing functionalities of the compressed suffix arrays. *J Algorithms* 48(2):294–313
- Schnieberger K, Hagmann J, Ossowski S, Warthmann N, Gesing S, Kohlbacher O, Weigel D (2009) Simultaneous alignment of short reads against multiple genomes. *Genome Biol* 10(9):R98
- Sirén J (2017) Indexing variation graphs. In: Proceedings of the 19th workshop on algorithm engineering and experiments (ALENEX), pp 13–27
- Sirén J, Välimäki N, Mäkinen V (2011) Indexing finite language representation of population genotypes. In: Proceedings of the 11th workshop on algorithms in bioinformatics (WABI), pp 270–281
- Sirén J, Välimäki N, Mäkinen V (2014) Indexing graphs for path queries with applications in genome research. *IEEE/ACM Trans Comput Biol Bioinform* 11(2):375–388
- Takabatake Y, Tabei Y, Sakamoto H (2014) Improved ESP-index: a practical self-index for highly repetitive texts. In: Proceedings of the 13th symposium on experimental algorithms (SEA), pp 338–350
- Takabatake Y, Nakashima K, Kuboyama T, Tabei Y, Sakamoto H (2016) siEDM: an efficient string index and search algorithm for edit distance with moves. *Algorithms* 9(2):26
- Takagi T, Goto K, Fujishige Y, Inenaga S, Arimura H (2017) Linear-size CDAWG: new repetition-aware indexing and grammar compression. In: Proceedings of the 24th symposium on string processing and information retrieval (SPIRE), pp 304–316
- Valenzuela D (2016) CHICO: a compressed hybrid index for repetitive collections. In: Proceedings of the 15th symposium on experimental algorithms (SEA), pp 326–338
- Valenzuela D, Mäkinen V (2017) CHIC: a short read aligner for pan-genomic references. Technical report 178129, bioRxiv.org
- Wandelt S, Leser U (2015) MRCSI: compressing and searching string collections with multiple references. *Proc VLDB Endowment* 8(5):461–472
- Wandelt S, Starlinger J, Bux M, Leser U (2013) RCSi: scalable similarity search in thousand(s) of genomes. *Proc VLDB Endowment* 6(13):1534–1545
- Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. *IEEE Trans Inf Theory* 23(3):337–343

Compressed Representations for Complex Networks

- ▶ (Web/Social) Graph Compression

Computational Needs of Big Data

- ▶ Parallel Processing with Big Data

Computer Architecture for Big Data

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

Big data hardware acceleration; Hardware considerations for big data

Definitions

How features of general-purpose computer architecture impact big-data applications and, conversely, how requirements of big data lead to the emergence of new hardware and architectural support.

Overview

Computer architecture (Parhami 2005) is a sub-discipline of computer science and engineering that is concerned with designing computing structures to meet application requirements

effectively, economically, reliably, and within prevailing technological constraints. In this entry, we discuss how features of general-purpose computer architecture impacts big-data applications and, conversely, how requirements of big data lead to the emergence of new hardware and architectural support.

Historical Trends in Computer Architecture

The von Neumann architecture for stored-program computers, with its single or unified memory, sometimes referred to as the Princeton architecture, emerged in 1945 (von Neumann 1945; von Neumann et al. 1947) and went virtually unchallenged for decades. It dominated the alternative Harvard architecture with separate program and data memories (Aiken and Hopper 1946) from the outset as the more efficient and versatile way of implementing digital computers.

As the workload for general-purpose computers began to change, adjustments in, and alternatives to, von Neumann architecture were proposed. Examples include de-emphasizing arithmetic operations in favor of data movement primitives, as seen in input/output and stream processors (Rixner 2001); introducing hardware aids for frequently used operations, as in graphic processing units or GPUs (Owens et al. 2008; Singer 2013); and adding special instructions for improved performance on multimedia workloads (Lee 1995; Yoon et al. 2001).

Recently, data-intensive applications necessitated another reassessment of the match between prevalent architectures and application requirements. The performance penalty of data having to be brought into the processor and sent back to memory through relatively narrow transfer channels was variously dubbed the “von Neumann bottleneck” (Markgraf 2007) and the “memory wall” (McKee 2004; Wulf and McKee 1995). Memory data transfer rates are measured in GB/s in modern machines, whereas the processing rates can be three or more decimal orders of magnitude higher.

The term “non-von” (Shaw 1982) was coined to characterize a large category of machines that relaxed one or more of the defining features of the von Neumann architecture, so as to alleviate some of the perceived problems. Use of cache memories (Smith 1982), often in multiple levels, eased the von Neumann bottleneck for a while, but the bottleneck reemerged, as the higher cache data transfer bandwidth became inadequate and applications that lacked or had relatively limited locality of reference emerged. Memory interleaving and memory-access pipelining, pioneered by IBM (Smotherman 2010) and later used extensively in Cray supercomputers, was the next logical step.

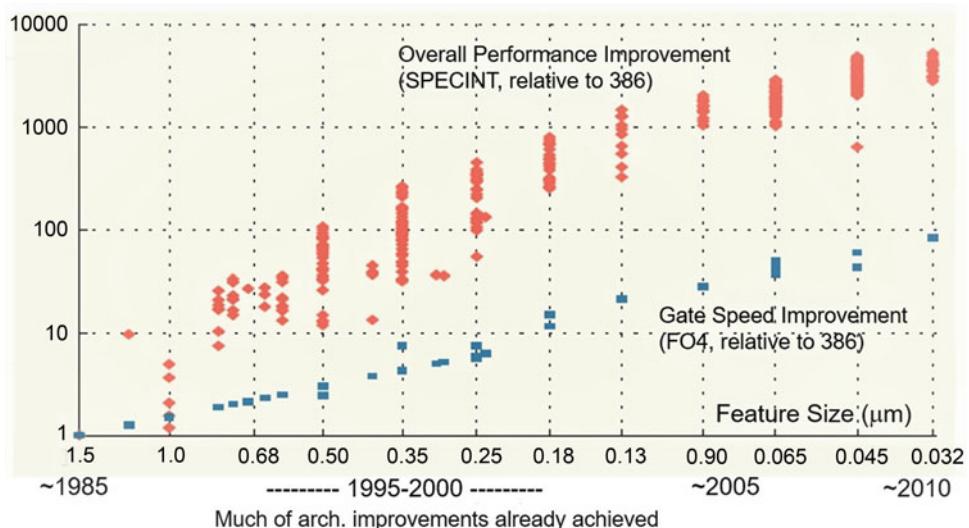
Extrapolating a bit from Fig. 1 (which covers the period 1985–2010), and using round numbers, the total effect of architectural innovations has been a 100-fold gain in performance, on top of another factor-of-100 improvement due to faster gates and circuits (Danowitz et al. 2012). Both growth rates in Fig. 1 show signs of slowing down, so that future gains to support the rising processing need of big data will have to come, at least in part, from other sources. In the technology arena, use of emerging technologies will provide some boost for specific applications. An intriguing option is resurrecting hybrid

digital/analog computing, which was sidelined long ago in favor of all-digital systems. Architecturally, specialization is one possible avenue for maintaining the performance growth rate, as are massively parallel and in-memory or near-memory computing.

How Big Data Affects Computer Architecture

The current age of big data (Chen and Zhang 2014; Hu et al. 2014) has once again exposed the von Neumann bottleneck, forcing computer architects to seek new solutions to the age-old problem, which has become much more serious. Processing speed continues to rise exponentially, while memory bandwidth increases at a much slower pace.

It is by now understood that big data is different from “lots of data.” It is sometimes defined in terms of the attributes of volume, variety, velocity, and veracity, known as the “4Vs” (or “5 Vs,” if we also include value). Dealing with big data requires big storage, big-data processing capability, and big communication bandwidth. The first two (storage and data processing) directly affect the architecture of the nodes holding



Computer Architecture for Big Data, Fig. 1 Technology advances and architectural innovations each contributed a factor of ~ 100 improvement in processor performance over three decades. (Danowitz et al. 2012)

and processing the data. The part of communication that is internode is separately considered in this encyclopedia. However, there is also the issue of intranode communication represented in buses and networks-on-chip that belong to our architectural discussion here.

In addition to data volume, the type of data to be handled is also changing from structured data, as reflected, for example, in relational databases, to semi-structured and unstructured data. While this change has some negative effects in terms of making traditional and well-understood database technologies obsolete, it also opens up the possibility of using scalable processing platforms made of commodity hardware as part of the cloud-computing infrastructure. Massive unstructured data sets can be stored in distributed file systems, such as the ones designed in connection with Hadoop (Shafer et al. 2010) or SQL/noSQL (Cattell 2011).

In addition to the challenges associated with rising storage requirements and data access bandwidth, the processing load grows with data volume because of various needs. These are:

- Encryption and decryption
- Compression and decompression
- Sampling and summarization
- Visualization and graphing
- Sorting and searching
- Indexing and query processing
- Classification and data mining
- Deduction and learning

The first four items above, which are different forms of data translation, are discussed in the next section. The other items, viz., data transformations, will be discussed subsequently.

Architectural Aids to Data Translations

Many important data translations are handled by endowing a general-purpose architecture with suitable accelerator units deployed as coprocessors. Such accelerators are ideally custom integrated circuits, whose designs are fully optimized

for their intended functions. However, in view of rapid advances in capabilities, performance, and energy efficiency of field-programmable gate arrays (Kuon et al. 2008), a vast majority of modern accelerators reported in the literature are built on FPGA circuits.

Accelerators for encryption and decryption algorithms have a long history (Bossuet et al. 2013). The binary choice of custom-designed VLSI or general-purpose processing for cryptographic computations has expanded to include a variety of intermediate solutions which include the use of FPGAs and GPUs. The best solution for an application domain depends not only on the required data rates and the crypto scheme, but also on power, area, and reliability requirements.

Data compression (Storer 1988) allows us to trade processing time and resources for savings in storage requirements. While any type of data can be compressed (e.g., text compression), massive sizes of video files make them a prime target for compression. With the emergence of video compression standards (Le Gall 1991), much effort has been expended to implement the standards on special-purpose hardware (Pirsch et al. 1995), offering orders of magnitude speed improvement over general-purpose programmed implementations.

Both sampling and summarization aim to reduce data volume while still allowing the operations of interest to be performed with reasonable precision. An alternative to post-collection reduction of data volume is to apply compression during data collection, an approach that in the case of sensor data collection is known as compressive sensing (Baraniuk 2007). Compressive sensing, when applicable, not only saves on processing time but also reduces transmission bandwidth and storage requirements. There are some mathematical underpinnings common to compressive sensing techniques, but at the implementation level, the methods and algorithms are by and large application-dependent.

Data visualization (Ward et al. 2010) refers to the production of graphical representation of data for better understanding of hidden structures and relationships. It may provide the only reasonable hope for understanding massive amounts of data,

although machine learning is a complementary and competing method of late. Several visualization accelerators were implemented in the late 1990s (e.g., Scott et al. 1998), but the modern trend is to use FPGA and cluster-based methods.

Architectural Aids to Data Transformations

Sorting is an extremely important primitive that is time-consuming for large data sets. It is used in a wide array of contexts, which includes facilitating subsequent searching operations. It can be accelerated in a variety of ways, from building more efficient data paths and memory access schemes, in order to make conventional sorting algorithms run faster, to the extreme of using hardware sorting networks (Mueller et al. 2012; Parhami 1999).

Indexing is one of the most important operations for large data sets, such as those maintained and processed by Google. Indexing and query processing have been targeted for acceleration within large-scale database implementations (Casper and Olukotun 2014; Govindaraju et al. 2004). Given the dominance of relational databases in numerous application contexts, a variety of acceleration methods have been proposed for operations on such databases (e.g., Bandi et al. 2004). Hardware components used in realizing such accelerators include both FPGAs and GPUs.

Hardware aids for classification are as diverse as classification algorithms and their underlying applications. A prime example in Internet routing is packet classification (Taylor 2005), which is needed when various kinds of packets, arriving at extremely high rates, must be separated for appropriate handling. Modern hardware aids for packet classification use custom arrays for pipelined network processing, content-addressable memories (Liu et al. 2010), GPUs (Owens et al. 2008), or tensor processing units (Sato et al. 2017). Data mining, the process of generating new information by examining large data sets, has also been targeted for acceleration (Sklyarov et al. 2015), and it can benefit from similar highly parallel processing approaches.

Also falling under such acceleration schemes are aids and accelerators for processing large graphs (Lee et al. 2017).

The earliest form of deduction engines were theorem provers. An important application of automatic deduction and proof is in hardware verification (Cyrluk et al. 1995). In recent years, machine learning has emerged as an important tool for improving the performance of conventional systems and for developing novel methods of tackling conceptually difficult problems. Game-playing systems (Chen 2016) constitute important testbeds for evaluating various approaches to machine learning and their associated hardware acceleration mechanisms. This is a field that has just started its meteoric rise and bears watching for future applications.

Memory, Processing, and Interconnects

In both general-purpose and special-purpose systems interacting with big data, the three interconnected challenges of providing adequate memory capacity, supplying the requisite processing power, and enabling high-bandwidth data movements between the various data-handling nodes must be tackled (Hilbert and Lopez 2011).

The memory problem can be approached using a combination of established and novel technologies, including nonvolatile RAM, 3D stacking of memory cells, processing in memory, content-addressable memory, and a variety of novel (nanoelectronics or biologically inspired) technologies. We won't dwell on the memory architecture in this entry, because the memory challenge is addressed in other articles (see the Cross-References).

Many established methods exist for increasing the data-handling capability of a processing node. The architectural nomenclature includes superscalar and VLIW organizations, collectively known as instruction-level parallelism (Rau and Fisher 1993), hardware multithreading (Eggers et al. 1997), multicore parallelism (Gepner and Kowalik 2006), domain-specific hardware accelerators (examples cited earlier in this entry),

transactional memory (Herlihy and Moss 1993), and SIMD/vector architectural or instruction-set extensions (Lee 1995; Yoon et al. 2001). A complete discussion of all these methods is beyond the scope of this entry, but much pertinent information can be found elsewhere in this encyclopedia.

Intranode communication is achieved through high-bandwidth bus systems (Hall et al. 2000) and, increasingly, for multicore processors and systems-on-chip, by means of on-chip networks (Benini and De Micheli 2002). Interconnection bandwidth and latency rank high, along with memory bandwidth, among hardware capabilities needed for effective handling of big-data applications, which are increasingly implemented using parallel and distributed processing. Considerations in this domain are discussed in the entry “Parallel Processing for Big Data.”

Future Directions

The field of computer architecture has advanced for several decades along the mainstream line of analyzing general applications and making hardware faster and more efficient in handling the common case while being less concerned with rare cases which have limited impact on performance. Big data both validates and challenges this assumption. It validates it in the sense that certain data-handling primitives arise in all contexts, regardless of the nature of the data or its volume. It challenges the assumption by virtue of the von-Neumann bottleneck or memory-wall notions discussed earlier. The age of big data will speed up the process of trickling down of architectural innovations from supercomputers, which have always led the way, into servers or even personal computers, which now benefit from parallel processing and, in some cases, massive parallelism.

Several studies have been performed about the direction of computer architecture in view of new application domains and technological developments in the twenty-first century (e.g., Ceze et al. 2016; Stanford 2012). Since much of the processing schemes for big data will be provided through

the cloud, directions of cloud computing and associated hardware acceleration mechanisms become relevant to our discussion here (Caulfield et al. 2016). Advanced graphics processors (e.g., Nvidia 2016) will continue to play a key role in providing the needed computational capabilities for data-intensive applications requiring heavy numerical calculations. Application-specific accelerators for machine learning (Sato et al. 2017), and, more generally, various forms of specialization, constitute another important avenue of architectural advances for the big-data universe.

Cross-References

- ▶ [Energy Implications of Big Data](#)
- ▶ [Parallel Processing with Big Data](#)
- ▶ [Storage Hierarchies for Big Data](#)
- ▶ [Storage Technologies for Big Data](#)

References

- Aiken HH, Hopper GM (1946) The automatic sequence controlled calculator – I. *Electr Eng* 65(8–9):384–391
- Bandi N, Sun C, Agrawal D, El Abbadi A (2004) Hardware acceleration in commercial databases: a case study of spatial operations. In: Proceedings of the international conference on very large data bases, Toronto, pp 1021–1032
- Baraniuk R (2007) Compressive sensing. *IEEE Signal Process Mag* 24(4):118–121
- Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. *IEEE Comput* 35(1):70–78
- Bossuet L, Grand M, Gaspar L, Fischer V, Gogniat G (2013) Architectures of flexible symmetric key crypto engines – a survey: from hardware coprocessor to multi-crypto-processor system on chip. *ACM Comput Surv* 45(4):41
- Casper J, Olukotun K (2014) Hardware acceleration of database operations. In: Proceedings of ACM/SIGDA international symposium on field-programmable gate arrays, Monterey, CA, pp 151–160
- Cattell R (2011) Scalable SQL and NoSQL data stores. *ACM SIGMOD Rec* 39(4):12–27
- Caulfield AM et al (2016) A cloud-scale acceleration architecture. In: Proceedings of 49th IEEE/ACM international symposium on microarchitecture, Orlando, FL, pp 1–13
- Ceze L, Hill MD, Wenisch TE (2016) Arch2030: a vision of computer architecture research over the next 15 years, Computing Community Consortium. <http://cra>.

- org/ccc/wp-content/uploads/sites/2/2016/12/15447-CC-C-ARCH-2030-report-v3-1-1.pdf
- Chen JX (2016) The evolution of computing: AlphaGo. *Comput Sci Eng* 18(4):4–7
- Chen CLP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inf Sci* 275:314–347
- Cyrluk D, Rajan S, Shankar N, Srivas MK (1995) Effective theorem proving for hardware verification. In: *Theorem provers in circuit design*. Springer, Berlin, pp 203–222
- Danowitz A, Kelley K, Mao J, Stevenson JP, Horowitz M (2012) CPU DB: recording microprocessor history. *Commun ACM* 55(4):55–63
- Eggers SJ, Emer JS, Levy HM, Lo JL, Stamm RL, Tullsen DM (1997) Simultaneous multithreading: a platform for next-generation processors. *IEEE Micro* 17(5): 12–19
- Gepner P, Kowalik MF (2006) Multi-core processors: new way to achieve high system performance. In: *Proceedings of IEEE international symposium on parallel computing in electrical engineering*, Bialystok, pp 9–13
- Govindaraju NK, Lloyd B, Wang W, Lin M, Manocha D (2004) Fast computation of database operations using graphics processors. In: *Proceedings of the ACM SIGMOD international conference on management of data*, Paris, pp 215–226
- Hall SH, Hall GW, McCall JA (2000) *High-speed digital system design: a handbook of interconnect theory and design practices*. Wiley, New York
- Herlihy M, Moss JEB (1993) Transactional memory: architectural support for lock-free data structures. In: *Proceedings of the international symposium on computer architecture*, San Diego, CA, pp 289–300
- Hilbert M, Lopez P (2011) The world's technological capacity to store, communicate, and compute information. *Science* 332:60–65
- Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. *IEEE Access* 2:652–687
- Kuon I, Tessier R, Rose J (2008) FPGA architecture: survey and challenges. *Found Trends Electron Des Autom* 2(2):135–253
- Le Gall D (1991) MPEG: a video compression standard for multimedia applications. *Commun ACM* 34(4): 46–58
- Lee RB (1995) Accelerating multimedia with enhanced microprocessors. *IEEE Micro* 15(2):22–32
- Lee J, Kim H, Yoo S, Choi K, Hofstee HP, Nam GJ, Nutter MR, Jamsek D (2017) ExtraV: boosting graph processing near storage with a coherent accelerator. *Proc VLDB Endowment* 10(12):1706–1717
- Liu AX, Meiners CR, Tornig E (2010) TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs. *IEEE/ACM Trans Networking* 18(2): 490–500
- Markgraf JD (2007) The von Neumann Bottleneck. Online source that is no longer accessible (will find a replacement for this reference during revisions)
- McKee SA (2004) Reflections on the memory wall. In: *Proceedings of the conference on computing frontiers*, Ischia, pp 162–167
- Mueller R, Teubner J, Alonso G (2012) Sorting networks on FPGAs. *Int J Very Large Data Bases* 21(1):1–23
- Nvidia (2016) Nvidia Tesla P100: infinite compute power for the modern data center – technical overview. On-line document. <http://images.nvidia.com/content/tesla/pdf/nvidia-teslap100-techoverview.pdf>. Accessed 18 Feb 2018
- Owens JD et al (2008) GPU computing. *Proc IEEE* 96(5):879–899
- Parhami B (1999) Chapter 7: Sorting networks. In: *Introduction to parallel processing: algorithms and architectures*. Plenum Press, New York, pp 129–147
- Parhami B (2005) *Computer architecture: from microprocessors to supercomputers*. Oxford University Press, New York
- Pirsich P, Demassieux N, Gehrke W (1995) VLSI architectures for video compression – a survey. *Proc IEEE* 83(2):220–246
- Rau BR, Fisher JA (1993) Instruction-level parallel processing: history, overview, and perspective. *J Supercomput* 7(1–2):9–50
- Rixner S (2001) Stream processor architecture. Kluwer, Boston
- Sato K, Young C, Patterson D (2017) An in-depth look at Google's first tensor processing unit, google cloud big data and machine learning blog, May 12. On-line document. <http://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>. Accessed 18 Feb 2018
- Scott ND, Olsen DM, Gannett EW (1998) An overview of the visualize FX graphic accelerator hardware. *Hewlett Packard J* 49:28–29
- Shafer J, Rixner S, Cox AL (2010) The Hadoop distributed filesystem: balancing portability and performance. In: *Proceedings of the IEEE international symposium on performance analysis of systems & software*, White Plains, NY, pp 122–133
- Shaw DE (1982) The non-von supercomputer, Columbia University technical report, on-line document. <http://academiccommons.columbia.edu/catalog/ac:140914>. Accessed 18 Feb 2018
- Singer G (2013) The history of the modern graphics processor, TechSpot on-line article. <http://www.techspot.com/article/650-history-of-the-gpu/>. Accessed 18 Feb 2018
- Sklyarov V et al (2015) Hardware accelerators for information retrieval and data mining. In: *Proceedings of the IEEE conference on information and communication technology research*, Bali, pp 202–205
- Smith AJ (1982) Cache memories. *ACM Comput Surv* 14(8):473–530
- Smotherman M (2010) IBM stretch (7030) – aggressive uniprocessor parallelism. On-line document. <http://people.cs.clemson.edu/~mark/stretch.html>. Accessed 18 Feb 2018
- Stanford University (2012) 21st century computer architecture: a community white paper, on-line document.

- http://csl.stanford.edu/~christos/publications/2012.21_stcenturyarchitecture.whitepaper.pdf. Accessed 18 Feb 2018
- Storer J (1988) Data compression. Computer Science Press, Rockville
- Taylor DE (2005) Survey and taxonomy of packet classification techniques. ACM Comput Surv 37(3):238–275
- von Neumann J (1945) First draft of a report on the EDVAC, University of Pennsylvania. On-line document. https://web.archive.org/web/20130314123032/http://qss.stanford.edu/~godfrey/vonNeumann/vne_dvac.pdf. Accessed 14 Feb 2018
- von Neumann J, Burks AW, Goldstine HH (1947) Preliminary discussion of the logical design of an electronic computing instrument. Institute for Advanced Study, Princeton
- Ward MO, Grinstein G, Keim D (2010) Interactive data visualization: foundations, techniques, and applications. CRC Press, Natick
- Wulf W, McKee S (1995) Hitting the wall: implications of the obvious. ACM Comput Archit News 23(1):20–24
- Yoon C-W, Woo R, Kook J, Lee S-J, Lee K, Yoo H-J (2001) An 80/20-MHz 160-mW multimedia processor integrated with embedded DRAM, MPEG-4 accelerator, and 3-D rendering engine for mobile applications. IEEE J Solid State Circuits 36(11):1758–1767

Definitions

Compression mechanisms reduce the storage cost of retained data. In the extreme case of data that must be retained indefinitely, the initial cost of performing the compression transformation can be amortized down to zero, since the savings in storage space continue to accrue without limit, albeit at decreasing rates as time goes by and disk storage becomes cheaper. A more typical scenario arises when a fixed data retention period must be supported, after which the stored data is no longer required; and when a certain level of access operations to the stored data can be expected, as part of a regulatory or compliance environment. In this second scenario, the *total cost of retention* (TCR) is a function of multiple competing factors, and the compression regime that provides the most compact storage might not be the one that provides the smallest TCR. This entry summarizes recent work in the area of cost models for data retention.

Computer Security

- ▶ [Big Data for Cybersecurity](#)

Computing Average Distance

- ▶ [Degrees of Separation and Diameter in Large Graphs](#)

Computing the Cost of Compressed Data

Alistair Moffat and Matthias Petri
 School of Computing and Information Systems,
 The University of Melbourne, Melbourne, VIC,
 Australia

Synonyms

[Data archiving](#); [Data compression](#); [Data retention](#)

Overview

Data compression techniques have received extensive study over more than six decades. Entropy coding mechanisms such as Huffman, arithmetic, and asymmetric numeral system (ANS) coding have been used to support a range of modeling approaches, including those based on implicit and explicit dictionaries, those based on statistical prediction, those based on grammar identification, and those based on the Burrows-Wheeler transform. Witten et al. (1999) and Moffat and Turpin (2002) provide details of many such combinations; the ANS mechanism is more recent (Duda 2009, 2013; Moffat and Petri 2017). Each possible arrangement of model and coder represents another option for use by practitioners, with a wide range of trade-offs possible. Compression systems are typically compared in two quite different ways, based on the size of the compressed data (the *effectiveness* of the approach) and based on the computational and memory resources required to attain the encoding

and decoding transformation (the *efficiency* of the approach). Moreover, it is often the case that effectiveness and efficiency are in tension – that the most effective techniques are also the ones that are least efficient and vice versa. For example, software implementations of compression tools often include an optional parameter (in several cases, “-1” to “-9”) to indicate the effort that should be used during encoding, with better compression resulting from greater effort; and principled trade-off mechanisms have also been proposed (Farruggia et al. 2014).

Hence, if a choice must be made between competing alternatives (compression programs or option settings) for use in some particular application or situation, a joint overall cost must be developed based on all relevant factors and used to measure the *total cost of retention*. This entry summarizes recent work by Liao et al. (2017) that describes such a cost model as a response to a *service-level agreement* and provides a synopsis of their key findings.

Data Storage and Retention

An archiving service stores different kinds of objects such as files, log entries, or variable-size data blobs. Adding objects to the storage system is generally referred to as a PUT request; and to retrieve objects, a GET request specifying one or more objects is issued to the storage system.

The objects stored by the archiving system can be viewed as a stream of bytes arriving at a constant rate when averaged over time. Liao et al. (2017) consider the case of data that arrives at an ingest rate of λ GiB/day, is subject to some obligatory minimum retention period of L_D days, and while retained must be available to GET requests assumed to occur at the rate of q queries per stored GiB per day. Each GET request for an object that was ingested during some particular day less than L_D days in the past is translated to a byte location within that day’s data, and a request for the corresponding object (some number of bytes or kibibytes) is then issued, so that the object can be retrieved and delivered. For example, log data or transac-

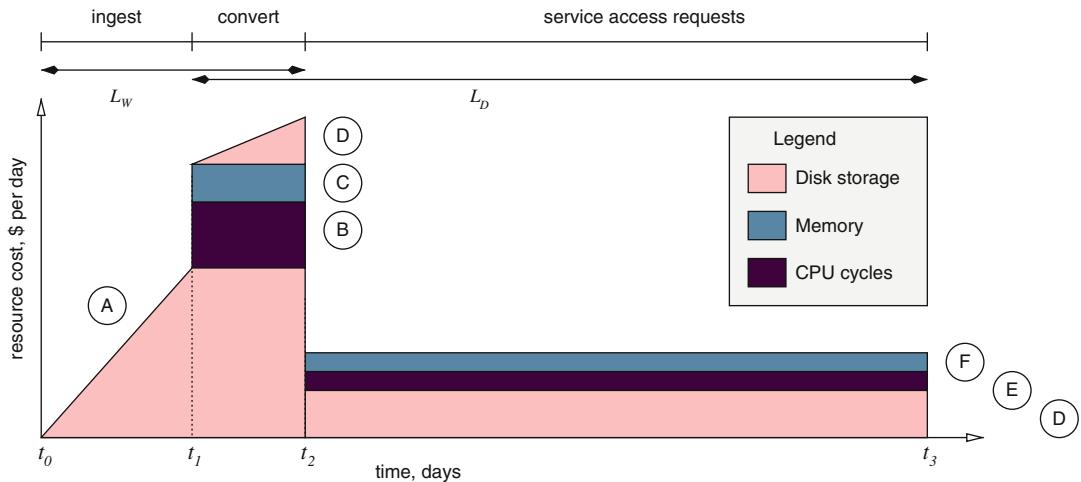
tion data might accumulate through some business process and be required to be retained for some minimum compliance period as part of a regulatory framework; and small fragments of that data might be required to be available on demand for audit or verification purposes. Such data retention regimes would typically be the subject of a *service-level agreement*, or SLA, including requirements in regard to the maximum delay L_W between ingest of any data and when it becomes available for querying operations, in regard to the maximum latency L_R permissible for any access operations, and in regard to other such performance and reliability requirements.

A cloud provider offering a data storage service naturally seeks to *minimize* the overall cost of providing the service within the constraints of the SLA. Compression is a key technology in this regard, since for long retention periods, the flag-fall cost of compressing the data can almost certainly be recouped via decreased storage costs – provided, that is, the required GET operations can also be supported within the constraints imposed by the SLA.

Cost Model for Data Retention

Liao et al. (2017) propose that the total cost of a storage system be subdivided into four categories broadly aligned with the components measured by most cloud system providers: (a) permanent storage, (b) working memory, (c) compute cycles, and (d) I/O operations. Each of these categories is then measured and billed over a specific time period, with the units of each category converted into a common currency of dollars. For example, consuming 50 GiB of permanent storage for 12 months might be billed at \$0.30 per GiB-month for a total storage cost of \$180; and the compute cycles necessary to support GET operations on that data might cost a further \$50 over the course of the year.

The service provider can be assumed to form data *bales* at daily or sub-daily intervals during the ingestion period, each bale assembled from multiple PUT operations. Once accumulated, the bale is then processed as a sequence of inde-



Computing the Cost of Compressed Data, Fig. 1 The bale-based data retention regime described by Liao et al. (2017) (Figure 2 of that paper, with copyright held by the original four authors). The six marked zones are: (A) cost of disk space for incoming data; (B) cost of CPU for index

construction, if applicable; (C) cost of memory space for index construction, if applicable; (D) cost of disk space for retained data; (E) cost of CPU for decoding and access operations during retention period; (F) cost of memory space during retention period, if applicable

pendently retrievable *blocks*, with the bale as a whole following the life cycle shown in Fig. 1. A generic compression algorithm, such as gzip, or a domain specific algorithm, such as MPEG, might be applied to each block, to reduce the amount of permanent storage required while that block is held through the retention period. Ideally a compression scheme which minimizes the overall balance between storage cost and resource expenditure should be chosen, subject to the constraints set by the SLA parameters. For example, the maximum write delay L_W and the maximum read latency L_R set constraints on the encoding and decoding speed of the compression algorithm and/or the size of the blocks that can be allowed. Similarly, the choice of permanent storage has implications on the types of compression algorithms which can comply with the SLA and also on the cost involved. For example, high-latency permanent storage devices will require a compression algorithm with fast decompression speed, in order to handle GET request within L_R .

That is, each possible algorithm requires a certain amount of compute and memory resources (zones (B) and (C) in Fig. 1) to convert incoming bales. After the bale is converted, it resides on secondary storage, incurring a constant storage

cost (D) for the remaining lifetime of the bale. To service GET requests arriving at an average rate of q queries per day per GiB of stored data, the system consumes additional compute and memory resources (E), (F) in order to decompress blocks. Additional IO costs may arise when transferring data to/from permanent storage and to/from the storage system itself.

Implications of the Cost Model

Liao et al. measure encoding and decoding speeds for compression mechanisms in four broad groups:

- No compression at all (denoted as method none);
- Fast encoding and decoding, based on an adaptive model and modest resource consumption, and with reasonably good compression effectiveness achieved at even relatively short block lengths (taking zlib, <https://github.com/madler/zlib>, as an exemplar);
- Excellent compression effectiveness when long blocks can be permitted, at the cost of more expensive encoding and decoding

- (taking `xz`, <http://tukaani.org/xz/>, as an exemplar); and
- RLZ, a semi-static mechanism designed for fast decoding and good compression regardless of block size but requiring a per-bale memory-resident dictionary throughout each bale’s retention period (Hoobin et al. 2011; Petri et al. 2015) (taking <https://github.com/mpetri/rlz-store> as an exemplar).

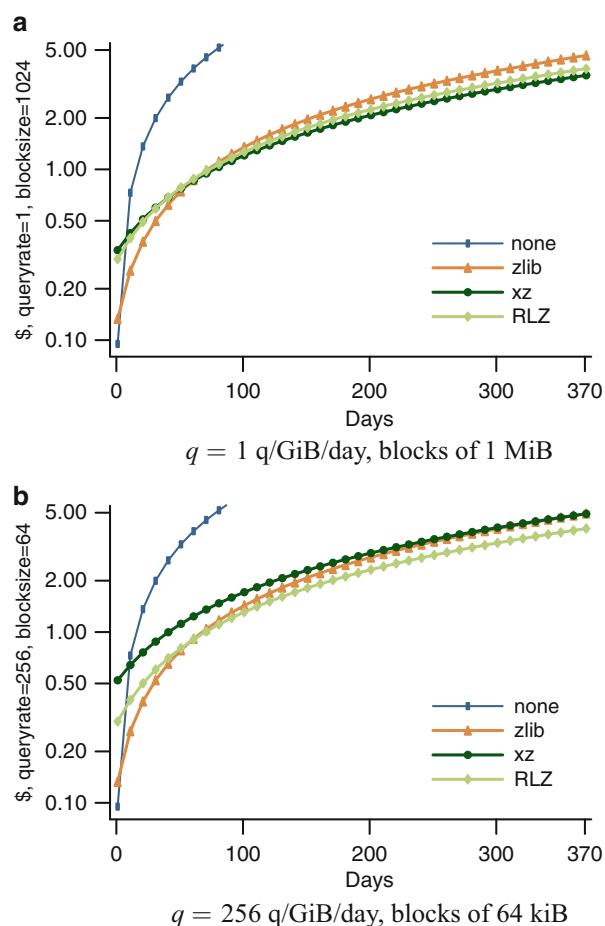
Liao et al. also provide compression effectiveness rates achieved for typical web data using a range of block sizes, highlighting the fact that for the adaptive zlib and xz approaches, compression effectiveness diminishes as blocks are made smaller.

Figure 2 takes that data and applies it to two different scenarios. In both graphs the total

retention cost (TRC, in dollars per 64 GiB bale) is plotted on the vertical axis as a function of cumulative retention time, shown on the horizontal axis, for durations from 1 day to a little over 1 year. In the first pane, a low query rate $q = 1$ query per day per stored GiB is assumed. In this arrangement, relatively large blocks of 1 MiB can be employed as the encoding unit, and as a result `xz` achieves excellent compression rates, close to those of the semi-static RLZ mechanism. Moreover, the 8 MiB allocation of memory required by RLZ (to be precise, 8 MiB in the configuration plotted; dictionary size is another dimension that affects the comparison) is a cost drain that is not recouped even though RLZ decoding is faster. Hence, for all but short retention durations, where the encoding speed of `zlib` is an advantage, `xz` provides the smallest TRC.

Computing the Cost of Compressed Data, Fig. 2

Total retention cost for two different scenarios: (a) a low daily access rate allows large blocks to be used, favoring the “very good but somewhat slow” compression regime `xz`; (b) a high query rate requires small blocks, favoring the semi-static RLZ mechanism. For short retention periods, `zlib` is the cheapest option in both scenarios; and storing the data uncompressed (method `none`) is never an attractive option



In the lower pane, the query rate is considerably increased, to $q = 256$ GET requests per stored GiB per day, and the block sizes need to shorten (the value $b = 64$ kB is used in the plot) so that decoding time doesn't swamp the other costs. Assuming that each GET request is to a different block, that query rate and block size mean that around 1.5% of the bale is decoded each day, and decoding throughput becomes a determining factor. Now RLZ's dictionary memory pays for itself, and it provides the best TRC, a combination of fast decoding and excellent compression effectiveness.

Based on the data they collected (including, as is also used in Figure 2, cost information from mid-2016 for a major cloud services provider), Liao et al. draw broad conclusions in regard to the efficacy of the various options:

- if the retention period is short, then zlib is the most economical choice, using a block size that decreases as the query rate increases;
- if the query rate is low, and if memory is expensive relative to secondary storage, then xz should be preferred; and
- if the query rate is high, and/or if the memory-to-disk cost ratio is more moderate, then RLZ-style approaches should be employed.

More generally, Liao et al. note that “thinking TRC” provides a framework in which compression improvements can be accurately measured and present one such approach that reduces RLZ-based retention costs by amortizing dictionary costs over a small number of consecutive bales.

Cross-References

- ▶ [Energy Implications of Big Data](#)
- ▶ [Hardware-Assisted Compression](#)
- ▶ [Storage Hierarchies for Big Data](#)

References

Duda J (2009) Asymmetric numeral systems. CoRR abs/0902.0271

- Duda J (2013) Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. CoRR abs/1311.2540
- Farruggia A, Ferragina P, Venturini R (2014) Bicriteria data compression: efficient and usable. In: Proceedings of the European symposium on algorithms (ESA), pp 406–417
- Hoobin C, Puglisi SJ, Zobel J (2011) Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. PVLDB 5(3):265–273
- Liao K, Moffat A, Petri M, Wirth A (2017) A cost model for long-term compressed data retention. In: Proceedings of the ACM international conference on web search and data mining (WSDM), pp 241–249
- Moffat A, Petri M (2017) ANS-based index compression. In: Proceedings of the ACM international conference on information and knowledge management (CIKM), pp 677–686
- Moffat A, Turpin A (2002) Compression and coding algorithms. Kluwer, Boston
- Petri M, Moffat A, Nagesh PC, Wirth A (2015) Access time tradeoffs in archive compression. In: Proceedings of the Asia information retrieval societies conference (AIRS), pp 15–28
- Witten IH, Moffat A, Bell TC (1999) Managing gigabytes: compressing and indexing documents and images. Morgan Kaufmann, San Francisco

Confidentiality

- ▶ [Security and Privacy in Big Data Environment](#)

Conflict-Free Replicated Data Types CRDTs

Nuno Preguiça¹, Carlos Baquero², and Marc Shapiro³

¹NOVA LINCS and DI, FCT, Universidade NOVA de Lisboa, Caparica, Portugal

²HASLab/INESC TEC and Universidade do Minho, Braga, Portugal

³Sorbonne Université, LIP6 and INRIA, Paris, France

Definitions

A conflict-free replicated data type (CRDT) is an abstract data type, with a well-defined interface, designed to be replicated at multiple pro-

cesses and exhibiting the following properties: (i) any replica can be modified without coordinating with other replicas and (ii) when any two replicas have received the same set of updates, they reach the same state, deterministically, by adopting mathematically sound rules to guarantee state convergence.

Overview

Internet-scale distributed systems often replicate data at multiple geographic locations to provide low latency and high availability, despite outages and network failures. To this end, these systems must accept updates at any replica and propagate these updates asynchronously to the other replicas. This approach allows replicas to temporarily diverge and requires a mechanism for merging concurrent updates into a common state. CRDTs provide a principled approach to address this issue.

As any abstract data type, a CRDT implements some given functionality and exposes a well-defined interface. Applications interact with the CRDT only through this interface. As CRDTs are designed to be replicated and to allow uncoordinated updates, a key aspect of a CRDT is its semantics in the presence of concurrency. The concurrency semantics defines what is the behavior of the object in the presence of concurrent updates, defining the state of the object for any given set of received updates.

An application developer uses the CRDT interface and concurrency semantics to reason about the behavior of her application in the presence of concurrent updates. A system developer creating a system that provides CRDTs needs to focus on another aspect of CRDTs: the synchronization model. The synchronization model defines the requirements that the system must meet so that CRDTs work correctly. We now detail each of these aspects independently.

Concurrency Semantics

The operations defined in a data type may intrinsically commute or not. Consider, for instance, a counter data type, a shared integer that supports

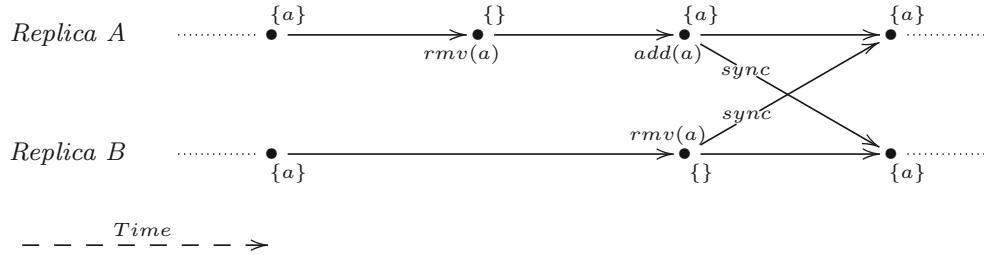
increment and decrement operations. As these operations commute (i.e., executing them in any order yields the same result), the counter data type naturally converges toward the expected result and reflects all executed operations reflects all executed operations.

Unfortunately, for most data types, this is not the case, and several concurrency semantics are reasonable, with different semantics being suitable for different applications. For instance, consider a shared memory cell supporting the assignment operation. If the initial value is 0, the correct outcome for concurrently assigning 1 and 2 is not well defined.

When defining the concurrency semantics, an important concept that is often used is that of the *happens-before* relation (Lamport 1978). In a distributed system, an event e_1 *happened-before* an event e_2 , $e_1 \prec e_2$, iff (i) e_1 occurred before e_2 in the same process; or (ii) e_1 is the event of sending message m , and e_2 is the event of receiving that message; or (iii) there exists an event e such that $e_1 \prec e$ and $e \prec e_2$. When applied to CRDTs, we can say that an update u_1 *happened-before* an update u_2 iff the effects of u_1 had been applied in the replica where u_2 was initially submitted.

As an example, if an event was *Alice reserved the meeting room*, it is relevant to know if that was known when *Bob reserved the meeting room* to determine if Alice should be given priority or if the two users concurrently tried to reserve the same room.

For instance, let us use *happened-before* to define the semantics of the *add-wins* set (also known as observed-remove set, OR-set (Shapiro et al. 2011)). Intuitively, in the *add-wins* semantics, in the presence of two operations that do not commute, a concurrent add and remove of the same element, the add wins leading to a state where the element belongs to the set. More formally, the set interface has two update operations: (i) $\text{add}(e)$, for adding element e to the set, and (ii) $\text{rmv}(e)$, for removing element e from the set. Given a set of update operations O that are related by the happens-before partial order \prec , the state of the set is defined as $\{e \mid \text{add}(e) \in O \wedge \exists \text{rmv}(e) \in O \cdot \text{add}(e) \prec \text{rmv}(e)\}$.



Conflict-Free Replicated Data Types CRDTs, Fig. 1 Run with an add-wins set

Figure 1 shows a run where an add-wins set is replicated in two replicas, with initial state $\{a\}$. In this example, in replica A, a is first removed and later added again to the set. In replica B, a is removed from the set. After receiving the updates from the other replica, both replicas end up with the element a in the set. The reason for this is that there is no $\text{rmv}(a)$ that happened after the $\text{add}(a)$ executed in replica A.

An alternative semantics based on the happens-before relation is *remove-wins*. Intuitively, in *remove-wins* semantics, in the presence of a concurrent add and remove of the same element, the remove wins leading to a state where the element is not in the set. More formally, given a set of update operations O , the state of the set is defined as: $\{e \mid \text{add}(e) \in O \wedge \forall \text{rmv}(e) \in O \cdot \text{rmv}(e) \prec \text{add}(e)\}$. In the previous example, after receiving the updates from the other replica, the state of both replicas would be the empty set, because there is no $\text{add}(a)$ that happened after the $\text{rmv}(a)$ in replica B.

Another relation that can be useful for defining the concurrency semantics is that of a total order among updates and particularly a total order that approximates wall-clock time. In distributed systems, it is common to maintain nodes with their physical clocks loosely synchronized. When combining the clock time with a site identifier, we have unique timestamps that are totally ordered. Due to the clock skew among multiple nodes, although these timestamps approximate an ideal global physical time, they do not necessarily respect the happens-before relation. This can be achieved by combining physical and logical clocks, as shown by Hybrid Logical Clocks (Kulkarni et al. 2014), or by only arbitrating

a wall-clock total order for the events that are concurrent under causality (Zawirski et al. 2016).

This relation allows to define the *last-writer-wins* semantics, where the value written by the last writer wins over the values written previously, according to the defined total order. More formally, with the set O of operations now totally ordered by $<$, the state of a *last-writer-wins* set would be defined as: $\{e \mid \text{add}(e) \in O \wedge \forall \text{rmv}(e) \in O \cdot \text{rmv}(e) < \text{add}(e)\}$. Returning to our previous example, the state of the replicas after the synchronization would include a if, according to the total order defined among the operations, the $\text{rmv}(a)$ of replica B is smaller than the $\text{add}(a)$ of replica A. Otherwise, the state would be the empty set.

We now briefly introduce the concurrency semantics proposed for several CRDTs.

Set

For a set CRDT, we have shown the difference between three possible concurrency semantics: *add-wins*, *remove-wins*, and *last-writer-wins*.

Register

A register CRDT maintains an opaque value and provides a single update operation that writes an arbitrary value: $\text{wr}(value)$. Two concurrency semantics have been proposed leading to two different CRDTs: the *multi-value* register and the *last-writer-wins* register. In the *multi-value* register, all concurrently written values are kept. In this case, the read operation returns the set of concurrently written values. Formally, the state of a multi-value register is defined as the multi-set: $\{v \mid \text{wr}(v) \in O \wedge \exists \text{wr}(u) \in O \cdot \text{wr}(v) \prec \text{wr}(u)\}$.

In the *last-writer-wins* register, only the value of the last write is kept, if any. Formally, the state of a last-writer-wins register can be defined as a set that is either empty or holds a single value: $\{v \mid \text{wr}(v) \in O \wedge \exists \text{wr}(u) \in O \cdot \text{wr}(v) < \text{wr}(u)\}$, assuming some initial write.

Counter

A counter CRDT maintains an integer and can be modified by update operations `inc` and `dec`, to increase and decrease by one unit its value, respectively (this can easily generalize to arbitrary amounts). As mentioned previously, as operations intrinsically commute, the natural concurrency semantics is to have a final state that reflects the effects of all registered operations. Thus, the result state is obtained by counting the number of increments and subtracting the number of decrements: $|\{\text{inc} \mid \text{inc} \in O\}| - |\{\text{dec} \mid \text{dec} \in O\}|$.

Now consider that we want to add a write operation $\text{wr}(n)$, to update the value of the counter to a given value. This opens two questions related with the concurrency semantics. First, what should be the final state when two concurrent write operations are executed? In this case, the last-writer-wins semantics would be simple (as maintaining multiple values, as in the multi-value register, is overly complex).

Second, what is the result when concurrent writes and `inc/dec` operations are executed? In this case, by building on the happens-before relation, we can define several concurrency semantics. One possibility is a *write-wins* semantics, where `inc/dec` operations have no effect when executed concurrently with the last write. Formally, for a given set O of updates that include at least a write operation, let v be the value in the last write, i.e., $\text{wr}(v) \in O \wedge \exists \text{wr}(u) \in O \cdot \text{wr}(v) < \text{wr}(u)$. The value of the counter would be $v + o$, with $o = |\{\text{inc} \mid \text{inc} \in O \wedge \text{wr}(v) < \text{inc}\}| - |\{\text{dec} \mid \text{dec} \in O \wedge \text{wr}(v) < \text{dec}\}|$ representing `inc/dec` operations that happened after the last write.

Other CRDTs

A number of other CRDTs have been proposed in the literature, including CRDTs for elementary

data structures, such as lists (Preguiça et al. 2009; Weiss et al. 2009; Roh et al. 2011), maps (Brown et al. 2014; Almeida et al. 2018), graphs (Shapiro et al. 2011), and more complex structures, such as JSON documents (Kleppmann and Beresford 2017). For each of these CRDTs, the developers have defined and implemented a type-specific concurrency semantics.

Synchronization Model

A replicated system needs to synchronize its replicas, by propagating and applying updates in every replica. There are two main approaches to propagate updates: state-based and operation-based replication.

In state-based replication, replicas synchronize by establishing bi-directional (or unidirectional) synchronization sessions, where both (one, resp.) replicas send their state to a peer replica. When a replica receives the state of a peer, it merges the received state with its local state. As long as the synchronization graph is connected, every update will eventually propagate to all replicas.

CRDTs designed for state-based replication define a merge function to integrate the state of a remote replica. It has been shown (Shapiro et al. 2011) that all replicas of a CRDT converge if (i) the states of the CRDT are partially ordered according to \leq forming a join semilattice; (ii) an operation modifies the state s of a replica by an inflation, producing a new state that is larger or equal to the original state according to \leq , i.e., for any operation m , $s \leq m(s)$; and (iii) the merge function computes the join (least upper bound) of two states, i.e., for states s, u , it computes $s \sqcup u$.

In operation-based replication, replicas converge by propagating operations to every other replica. When an operation is received in a replica, it is applied to the local replica state. Besides requiring that every operation is reliably delivered to all replicas, e.g., by using some reliable multicast communication subsystem, some systems may require operations to be delivered according to some specific order, with causal order being the most common.

CRDTs designed for operation-based replication must define, for each operation, a generator

and an effector function. The generator function executes in the replica where the operation is submitted, the source replica, it has no side effects and generates an effector that encodes the side effects of the operation. In other words, the effector is a closure created by the generator depending on the state of the origin replica. The effector operation must be reliably executed in all replicas, where it updates the replica state. Shapiro et al. (2011) have shown that if effector operations are delivered in causal order, replicas will converge to the same state if concurrent effector operations commute. If effector operations may be delivered without respecting causal order, then all effector operations must commute. Most operation-based CRDT designs require causal delivery.

Alternative models: When operations modify only part of the state, propagating the complete state for synchronization to a remote replica is inefficient, as the remote replica already knows most of the state. Delta-state CRDTs (Almeida et al. 2018) address this issue by propagating only delta mutators, which encode the changes that have been made to a replica since the last communication. The first time a replica communicates with some other replica, the full state needs to be propagated. This can be improved by using a state summary (e.g., version vector) for computing and sending only the deltas in the first communication also, as shown in big delta-state CRDTs (van der Linde et al. 2016), typically at the cost of storing more metadata in the CRDT state. Another improvement is to compute digests that help determine which parts of a remote state are needed, avoiding shipping full states (Enes 2017).

In the context of operation-based replication, effector operations should be applied immediately to the source replica. However, propagation to other replicas can be deferred for some period and effectors stored in an outbound log, presenting an opportunity to compress the log by rewriting some operations – e.g., two `add(1)` operations in a counter can be converted in a `add(2)` operation. This mechanism has been used by Cabrita and Preguiça (2017). Delta mutators can also be seen as a compressed representation of a log of operations.

Operation-based CRDTs require executing a generator function against the replica state to compute an effector operation. In some scenarios, this may introduce an unacceptable delay for propagating an operation. Pure operation-based CRDTs (Baquero et al. 2014) address this issue by allowing the original operations to be propagated to all replicas, typically at the cost of more complex operations and of having to store more metadata in the CRDT state.

C

Key Research Findings

Preservation of Sequential Semantics

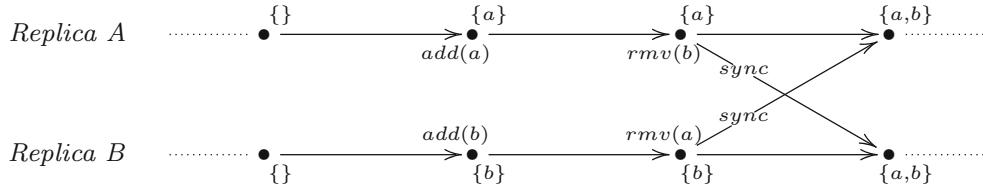
When modeling an abstract data type that has an established semantics under sequential execution, CRDTs should preserve that semantics. For instance, CRDT sets should ensure that if the last operation in a sequence of operations to a set added a given element, then a query operation immediately after that one will show the element to be present on the set. Conversely, if the last operation removed an element, then a subsequent query should not show its presence.

Sequential execution can occur even in distributed settings if synchronization is frequent. An instance can be updated in replica A, merged into another replica B and updated there, and merged back into replica A before A tries to update it again. In this case, we have a sequential execution, even though updates have been executed in different replicas.

Historically, not all CRDT designs have met this property. The *two-phase-set* CRDT (2PSet) does not allow re-adding an element that was removed, and thus it breaks the common sequential semantics. Later CRDT set designs, such as *add-wins* and *remove-wins* sets, do preserve the original sequential semantics while providing different concurrency semantics.

Extended Behavior Under Concurrency

Some CRDT designs handle concurrent operations by arbitrating a given sequential ordering to accommodate concurrent execution. For example, the state of a *last-writer-wins* set replica can be explained by a sequential execution of the op-



Conflict-Free Replicated Data Types CRDTs, Fig. 2 Add-wins set run showing that there might be no sequential execution of operations that explains CRDT behavior

erations according to the LWW total order used. When operations commute, such as in counters, there might even be several sequential executions that explain a given state.

Not all CRDTs need or can be explained by sequential executions. The add-wins set is an example of a CRDT where there might be no sequential execution of operations to explain the state observed, as Fig. 2 shows. In this example, the state of the set after all updates propagate to all replicas includes a and b , but in any sequential extension of the causal order, a remove operation would always be the last operation, and consequently the removed element could not belong to the set.

Some other CRDTs can exhibit states that are only reached when concurrency does occur. An example is the *multi-value register*. If used sequentially, sequential semantics is preserved, and a read will show the outcome of the most recent write in the sequence. However, if two or more values are written concurrently, the subsequent read will show all those values (as the *multi-value* name implies), and there is no sequential execution that can explain this result. We also note that a follow-up write can overwrite both a single value and multiple values.

Guarantees and Limitations

An important property of CRDTs is that an operation can always be accepted at any given replica and updates are propagated asynchronously to other replicas. In the CAP theorem framework (Brewer 2010; Gilbert and Lynch 2002), the CRDT conflict-free approach favors availability over consistency when facing communication disruptions. This leads to resilience to network failure and disconnection,

since no prior coordination with other replicas is necessary before accepting an operation. Furthermore, operations can be accepted with minimal user perceived latency since they only require local durability. By eschewing global coordination, replicas evolve independently, and reads will not reflect operations accepted in remote replicas that have not yet been propagated to the local replica.

In the absence of global coordination, session guarantees (Terry et al. 1994) specify what the user applications can expect from their interaction with the system's interface. Both state-based CRDTs and operation-based CRDTs when supported by reliable causal delivery provide per-object causal consistency. Thus, in the context of a given replicated object, the traditional session guarantees are met. CRDT-based systems that lack transactional support can enforce system-wide causal consistency, by integrating multiple objects in a single map/directory object (Almeida et al. 2018). Another alternative is to use mergeable transactions to read from a causally consistent database snapshot and to provide write atomicity (Preguiça et al. 2014).

Some operations cannot be expressed in a conflict-free framework and will require global agreement. As an example, in an auction system, bids can be collected under causal consistency, and a new bid will only have to increase the offer with respect to bids that are known to causally precede it. However, closing the auction and selecting a single winning bid will require global agreement. It is possible to design a system that integrates operations with different coordination requirements and only resorts to global agreement when necessary (Li et al. 2012; Sovran et al. 2011).

Some global invariants, which usually are enforced with global coordination, can be enforced in a conflict-free manner by using escrow techniques (O’Neil 1986) that split the available resources among the different replicas. For instance, the Bounded Counter CRDT (Balegas et al. 2015b) defines a counter that never goes negative, among assigning to each replica a reserve of allowed decrements under the condition that the sum of all allowed decrements does not exceed the value of the counter. As long as its reserve is not exhausted, a replica can accept decrements without coordinating with other replicas. After a replica exhausts its reserve, a new decrement will either fail or require synchronizing with some replica that still can decrement. This technique uses point-to-point coordination and can be generalized to enforce other system-wide invariants (Balegas et al. 2015a)

Examples of Applications

CRDTs have been used in a large number of distributed systems and applications that adopt weak consistency models. The adoption of CRDTs simplifies the development of these systems and applications, as CRDTs guarantee that replicas converge to the same state when all updates are propagated to all replicas. We can group the systems and applications that use CRDTs into two groups: storage systems that provide CRDTs as their data model and applications that embed CRDTs to maintain their internal data.

CRDTs have been integrated in several storage systems that make them available to applications. An application uses these CRDTs to store its data, being the responsibility of the storage systems to synchronize the multiple replicas. The following commercial systems use CRDTs: Riak (Developing with Riak KV Data Types <http://docs.basho.com/riak/kv/2.2.3/developing/data-types/>), Redis (Biyikoglu 2017), and Akka (Akka Distributed Data: <https://doc.akka.io/docs/akka/2.5.4/scala/distributed-data.html>).

A number of research prototypes have also used CRDTs, including Walter (Sovran et al. 2011),

SwiftCloud (Preguiça et al. 2014), and Antidote (Antidote: <http://antidotedb.org>) (Akkoorath et al. 2016).

CRDTs have also been embedded in multiple applications. In this case, developers either used one of the available CRDT libraries, implemented themselves some previously proposed design, or designed new CRDTs to meet their specific requirements. An example of this latter use is Roshi (Roshi is a large-scale CRDT set implementation for timestamped events <https://github.com/soundcloud/rosi>.), a LWW-element-set CRDT used for maintaining an index in SoundCloud stream.

C

Future Directions of Research

Scalability

In order to track concurrency and causal predecessors, CRDT implementations often store metadata that grows linearly with the number of replicas (Charron-Bost 1991). While global agreement suffers from greater scalability limitations since replicas must coordinate to accept each operation, the metadata cost from causality tracking can limit the scalability of CRDTs when aiming for more than a few hundred replicas. A large metadata footprint can also impact the computation time of local operations and will certainly impact the required storage and communication.

Possible solutions can be sought in more compact causality representations when multiple replicas are synchronized among the same nodes (Malkhi and Terry 2007; Preguiça et al. 2014; Gonçalves et al. 2017) or by hierarchical approaches that restrict all to all synchronization and enable more compact mechanisms (Almeida and Baquero 2013).

Reversible Computation

Nontrivial Internet services require the composition of multiple subsystems, to provide storage, data dissemination, event notification, monitoring, and other needed components. When composing subsystems, which can fail independently or simply reject some operations, it is useful

to provide a CRDT interface that undoes previously accepted operations. Another scenario that would benefit from undo is collaborative editing of shared documents, where undo is typically a feature available to users.

Undoing an increment on a counter CRDT can be achieved by a decrement. Logoot-Undo (Weiss et al. 2010) proposes a solution for undoing (and redoing) operations for a sequence CRDT used for collaborative editing. However, providing a uniform approach to undoing, reversing, operations over the whole CRDT catalog is still an open research direction. The support of undo is also likely to limit the level of compression that can be applied to CRDT metadata.

Security

While access to a CRDT-based interface can be restricted by adding authentication, any accessing client has the potential to issue operations that can interfere with the other replicas. For instance, delete operations can remove all existing state. In state-based CRDTs, replicas have access to state that holds a compressed representation of past operations and metadata. By manipulation of this state and synchronizing to other replicas, it is possible to introduce significant attacks to the system operation and even its future evolution.

Applications that store state on third-party entities, such as in cloud storage providers, might not trust the provider and choose end-to-end encryption of the exchanged state. This, however, would require all processing to be done at the edge, under the application control. A research direction would be to allow some limited form of computation, such as merging state, over information whose content is subject to encryption. Potential techniques, such as homomorphic encryption, are likely to pose significant computational costs. An alternative is to execute operations on encrypted data without disclosing it, relying on specific hardware support, such as Intel SGX and ARM TrustZone.

Nonuniform Replicas

The replication of CRDTs typically assumes that eventually all replicas will reach the same state, storing exactly the same data. However, depend-

ing on the read operations available in the CRDT interface, it might not be necessary to maintain the same state in all replicas. For example, an object that has a single read operation returning the top-K elements added to the object only needs to maintain those top-K elements in every replica. The remaining elements are necessary if a remove operation is available, as one of the elements not in the top needs to be promoted when a top element is removed. Thus, each replica can maintain only the top-K elements and the elements added locally.

This replication model is named nonuniform replication (Cabrita and Preguiça 2017) and can be used to design CRDTs that exhibit important storage and bandwidth savings when compared with alternatives that keep all data in all replicas. Although it is clear that this model cannot be used for all data types, several useful CRDT designs have been proposed, including top-K, top-Sum, and histogram. To understand what data types can adopt this model and how to explore it in practice is an open research question.

Verification

An important aspect related with the development of distributed systems that use CRDTs is the verification of the correctness of the system. This involves not only verifying the correctness of CRDT designs but also the correctness of the system that uses CRDTs. A number of works have addressed these issues.

Regarding the verification of the correctness of CRDTs, several approaches have been taken. The most commonly used approach is to have proofs when designs are proposed or to use some verification tools for the specific data type, such as TLA (Lamport 1994) or Isabelle (Isabelle: <http://isabelle.in.tum.de/>). There have also been some works that proposed general techniques for the verification of CRDTs (Burckhardt et al. 2014; Zeller et al. 2014; Gomes et al. 2017), which can be used by CRDT developers to verify the correctness of their designs. Some of these works (Zeller et al. 2014; Gomes et al. 2017) include specific frameworks that help the developer in the verification process.

A number of other works have proposed techniques to verify the correctness of distributed systems that use CRDTs (Gotsman et al. 2016; Zeller 2017; Balegas et al. 2015a). These works typically require the developer to specify the properties that the distributed system must maintain and a specification of the operations in the system (that is independent of the actual code of the system). Despite these works, the verification of the correctness of CRDT designs and of systems that use CRDTs, how these verification techniques can be made available to programmers, and how to verify the correctness of implementations remain an open research problem.

Acknowledgements This work was partially supported by NOVA LINCS (UID/CEC/04516/2013), EU H2020 LightKone project (732505), and SMILES line in project TEC4Growth (NORTE-01-0145-FEDER-000020).

References

- Akkoorath DD, Tomsic AZ, Bravo M, Li Z, Crain T, Bieniusa A, Preguiça N, Shapiro M (2016) Cure: strong semantics meets high availability and low latency. In: Proceedings of the 2016 IEEE 36th international conference on distributed computing systems (ICDCS), pp 405–414. <https://doi.org/10.1109/ICDCS.2016.98>
- Almeida PS, Baquero C (2013) Scalable eventually consistent counters over unreliable networks. CoRR abs/1307.3207. <http://arxiv.org/abs/1307.3207>, 1307.3207
- Almeida PS, Shoker A, Baquero C (2018) Delta state replicated data types. J Parallel Distrib Comput 111:162–173. <https://doi.org/10.1016/j.jpdc.2017.08.003>
- Balegas V, Duarte S, Ferreira C, Rodrigues R, Preguiça NM, Najafzadeh M, Shapiro M (2015a) Putting consistency back into eventual consistency. In: Réveillère L, Harris T, Herlihy M (eds) Proceedings of the tenth European conference on computer systems, EuroSys 2015, Bordeaux. ACM, pp 6:1–6:16. <https://doi.org/10.1145/2741948.2741972>
- Balegas V, Serra D, Duarte S, Ferreira C, Shapiro M, Rodrigues R, Preguiça NM (2015b) Extending eventually consistent cloud databases for enforcing numeric invariants. In: 34th IEEE symposium on reliable distributed systems, SRDS 2015, Montreal. IEEE Computer Society, pp 31–36. <https://doi.org/10.1109/SRDS.2015.32>
- Baquero C, Almeida PS, Shoker A (2014) Making operation-based CRDTs operation-based. In: Proceedings of the first workshop on principles and practice of eventual consistency, PaPEC’14. ACM, New York, pp 7:1–7:2. <https://doi.org/10.1145/2596631.2596632>
- Biyikoglu C (2017) Under the hood: Redis CRDTs (conflict-free replicated data types). Online <https://goo.gl/tGqU7h>. Accessed 24 Nov 2017
- Brewer E (2010) On a certain freedom: exploring the CAP space, invited talk at PODC 2010, Zurich
- Brown R, Cribbs S, Meiklejohn C, Elliott S (2014) Riak DT map: a Composable, convergent replicated dictionary. In: Proceedings of the first workshop on principles and practice of eventual consistency, PaPEC’14. ACM, New York, pp 1:1–1:1. <https://doi.org/10.1145/2596631.2596633>
- Burckhardt S, Gotsman A, Yang H, Zawirski M (2014) Replicated data types: specification, verification, optimality. In: Proceedings of the 41st ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL’14. ACM, New York, pp 271–284. <https://doi.org/10.1145/2535838.2535848>
- Cabrita G, Preguiça N (2017) Non-uniform replication. In: Proceedings of the 21th international conference on principles of distributed systems, OPODIS 2017, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, LIPIcs
- Charron-Bost B (1991) Concerning the size of logical clocks in distributed systems. Inf Process Lett 39(1):11–16. [https://doi.org/10.1016/0020-0190\(91\)90055-M](https://doi.org/10.1016/0020-0190(91)90055-M)
- Enes V (2017) Efficient Synchronization of State-based CRDTs. Master’s thesis, Universidade do Minho. <http://vitorenedesduarte.github.io/page/other/msc-thesis.pdf>
- Gilbert S, Lynch N (2002) Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59. <https://doi.org/10.1145/564585.564601>
- Gomes VBF, Kleppmann M, Mulligan DP, Beresford AR (2017) Verifying strong eventual consistency in distributed systems. Proc ACM Program Lang 1(OOPSLA):109:1–109:28. <https://doi.org/10.1145/3133933>
- Gonçalves RJT, Almeida PS, Baquero C, Fonte V (2017) DottedDB: anti-entropy without Merkle trees, deletes without tombstones. In: Proceedings of the 2017 IEEE 36th symposium on reliable distributed systems (SRDS), pp 194–203. <https://doi.org/10.1109/SRDS.2017.28>
- Gotsman A, Yang H, Ferreira C, Najafzadeh M, Shapiro M (2016) ‘cause i’m strong enough: reasoning about consistency choices in distributed systems. In: Proceedings of the 43rd annual ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL’16. ACM, New York, pp 371–384. <https://doi.org/10.1145/2837614.2837625>
- Kleppmann M, Beresford AR (2017) A conflict-free replicated JSON datatype. IEEE Trans Parallel Distrib Syst 28(10):2733–2746. <https://doi.org/10.1109/TPDS.2017.2697382>
- Kulkarni SS, Demirbas M, Madappa D, Avva B, Leone M (2014) Logical physical clocks. In: Aguilera MK, Querzoni L, Shapiro M (eds) Principles of distributed systems – 18th international conference, OPODIS

- 2014, Cortina d'Ampezzo. Proceedings. Lecture notes in computer science, vol 8878. Springer, pp 17–32. https://doi.org/10.1007/978-3-319-14472-6_2
- Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. <https://doi.org/10.1145/359545.359563>
- Lamport L (1994) The temporal logic of actions. ACM Trans Program Lang Syst 16(3):872–923. <https://doi.org/10.1145/177492.177726>
- Li C, Porto D, Clement A, Gehrke J, Preguiça N, Rodrigues R (2012) Making geo-replicated systems fast as possible, consistent when necessary. In: Proceedings of the 10th USENIX conference on operating systems design and implementation, OSDI'12. USENIX Association, Berkeley, pp 265–278. <http://dl.acm.org/citation.cfm?id=2387880.2387906>
- Malkhi D, Terry DB (2007) Concise version vectors in winfs. Distrib Comput 20(3):209–219. <https://doi.org/10.1007/s00446-007-0044-y>
- O'Neil PE (1986) The escrow transactional method. ACM Trans Database Syst 11(4):405–430. <https://doi.org/10.1145/7239.7265>
- Preguiça N, Marques JM, Shapiro M, Letia M (2009) A commutative replicated data type for cooperative editing. In: Proceedings of the 2009 29th IEEE international conference on distributed computing systems, ICDCS'09. IEEE Computer Society, Washington, DC, pp 395–403. <https://doi.org/10.1109/ICDCS.2009.20>
- Preguiça NM, Zawirski M, Bieniusa A, Duarte S, Ballegas V, Baquero C, Shapiro M (2014) Swiftcloud: fault-tolerant geo-replication integrated all the way to the client machine. In: 33rd IEEE international symposium on reliable distributed systems workshops, SRDS workshops 2014, Nara. IEEE Computer Society, pp 30–33. <https://doi.org/10.1109/SRDSW.2014.33>
- Roh HG, Jeon M, Kim JS, Lee J (2011) Replicated abstract data types: building blocks for collaborative applications. J Parallel Distrib Comput 71(3):354–368. <https://doi.org/10.1016/j.jpdc.2010.12.006>
- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) Conflict-free replicated data types. In: Proceedings of the 13th international conference on stabilization, safety, and security of distributed systems, SSS'11. Springer, Berlin/Heidelberg, pp 386–400. <http://dl.acm.org/citation.cfm?id=2050613.2050642>
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 385–400. <https://doi.org/10.1145/2043556.2043592>
- Terry DB, Demers AJ, Petersen K, Spreitzer M, Theimer M, Welch BB (1994) Session guarantees for weakly consistent replicated data. In: Proceedings of the third international conference on parallel and distributed information systems, PDIS'94, Austin. IEEE Computer Society, pp 140–149. <https://doi.org/10.1109/PDIS.1994.331722>
- van der Linde A, Leitão JA, Preguiça N (2016) Δ -CRDTs: making δ -CRDTs delta-based. In: Proceedings of the 2nd workshop on the principles and practice of consistency for distributed data, PaPoC'16. ACM, New York, pp 12:1–12:4. <https://doi.org/10.1145/2911151.2911163>
- Weiss S, Urso P, Molli P (2009) Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks. In: Proceedings of the 2009 29th IEEE international conference on distributed computing systems, ICDCS'09. IEEE Computer Society, Washington, DC, pp 404–412. <https://doi.org/10.1109/ICDCS.2009.75>
- Weiss S, Urso P, Molli P (2010) Logoot-undo: distributed collaborative editing system on p2p networks. IEEE Trans Parallel Distrib Syst 21(8):1162–1174. <https://doi.org/10.1109/TPDS.2009.173>
- Zawirski M, Baquero C, Bieniusa A, Preguiça N, Shapiro M (2016) Eventually consistent register revisited. In: Proceedings of the 2nd workshop on the principles and practice of consistency for distributed data, PaPoC'16. ACM, New York, pp 9:1–9:3. <https://doi.org/10.1145/2911151.2911157>
- Zeller P (2017) Testing properties of weakly consistent programs with repliss. In: Proceedings of the 3rd international workshop on principles and practice of consistency for distributed data, PaPoC'17. ACM, New York, pp 3:1–3:5. <https://doi.org/10.1145/3064889.3064893>, <https://dl.acm.org/authorize?N37605>
- Zeller P, Bieniusa A, Poetzsch-Heffter A (2014) Formal specification and verification of CRDTs. In: Formal techniques for distributed objects, FORTE 2014. Lecture notes in computer science. Springer, pp 33–48

Conformance Checking

Jorge Munoz-Gama

Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Synonyms

[Business process conformance checking](#)

Definitions

Given an event log and a process model from the same process, conformance checking compares the recorded event data with the model to

identify commonalities and discrepancies. The conformance between a log and model can be quantified with respect to different quality dimensions: *fitness*, *precision*, and *generalization*.

Overview

Conformance checking compares an event log with a process model of the same process (Munoz-Gama 2016). An event log is composed of a series of log traces where each log trace relates to the sequence of observed events of a process instance, i.e., a case. An event can be related to a particular activity in the process but can also record many other process information such as time stamp, resource, and cost. In a real-life context, event logs can be extracted from Process-Aware Information Systems (PAIS) such as workflow management (WFM) systems, business process management (BPM) systems, or typical relational databases, such as SAP database. Similarly, process models can often be extracted from the organization's information systems. These can be normative models that the organization uses to manage their process, or descriptive, created by hand or automatically discovered to gain insight into their processes (van der Aalst 2013).

Depending on the nature of the model, discrepancies between the log and model can have different interpretations (van der Aalst 2016). For a normative model, deviations indicate violations of imposed constraints. For example, a banking process may require the processing and approval of a loan to be done by different employees to avoid the risk of misconduct (four-eyes principle). Clearly, conformance checking between an event log of the handled loan applications and the process model can be applied to assess compliance. On the other hand, for a descriptive model, deviations indicate that the model is not fully capturing all the observed behavior in the log. For example, process analysts might perform conformance checking on the models discovered by different process discovery algorithms before selecting the ones that are of sufficient quality for further analysis.

To illustrate conformance checking, a simple process is introduced. Figure 1 shows a doctoral scholarship application process in an informal modeling notation. This process consists of eight activities: *Start Processing*, *Evaluate Project*, *Evaluate Academic Record*, *Evaluate Advisor CV*, *Final Evaluation*, *Accept*, *Reject*, and *Notify Results*. To begin the process, an applicant has to submit their academic record, their advisor's CV, and a description of their proposed project. Once the required documents are received, the committee would begin by evaluating the submitted documents. As shown by the AND gateway, the committee can choose to evaluate the three documents in any order. Following the preliminary evaluation, a final evaluation is done to consolidate the previous results. This leads to either the acceptance or rejection of the application. Finally, the applicant is notified of the result. An example of log trace corresponding to an accepted application could be $\langle \text{Start Processing}, \text{Evaluate Project}, \text{Evaluate Academic Record}, \text{Evaluate Advisor CV}, \text{Final Evaluation}, \text{Accept}, \text{Notify Results} \rangle$.

Dimensions of Conformance

Through conformance checking, commonalities and discrepancies between a log and model are quantified. One simple idea would be to consider that a log and model are conforming with each other if the observed behavior in the log is captured by the model. This means that a log and model are perfectly conforming if all the log traces can be *fitted* to the model. However, this can be easily achieved with a model that allows any behavior. Such models do not provide much information to the data analyst about the process. This shows that there is a need to consider conformance with respect to different dimensions.

Currently, conformance is generally considered with respect to three dimensions – *fitness*, *precision*, and *generalization*.

Fitness relates to how well a model and log fit each other. A log trace perfectly *fits* the model if it can be replayed onto the model and corresponds to a complete model trace. For example, $\langle \text{Start Processing}, \text{Evaluate Project}, \text{Evaluate Academic Record} \rangle$

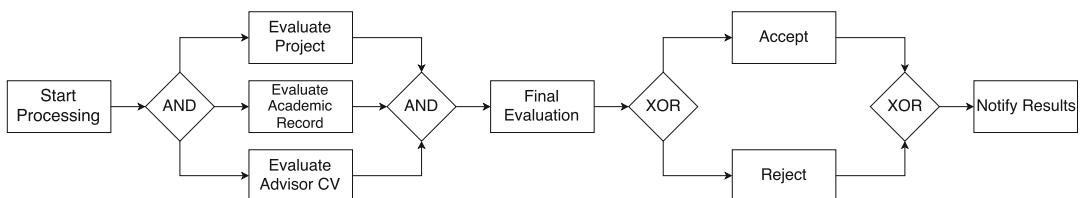
Record, Evaluate Advisor CV, Final Evaluation, Accept, Notify Results} perfectly fits the model in Fig. 1 since each of the observed steps can be sequentially replayed at the model, and the trace corresponds to a particular possible way to execute the process model. However, the trace *(Start Processing, Evaluate Project, Evaluate Academic Record, Final Evaluation, Reject, Notify Results)* does not fit the model because the advisor's CV (*Evaluate Advisor CV*) is never evaluated. This suggests that the corresponding application has been rejected without proper evaluation.

Precision relates to a model's ability to capture the observed behavior without allowing unseen behavior. It is not enough to have a model that is perfectly fitting with the log since this can be easily achieved with a model that permits any behavior. Consider the "flower" model in Fig. 2; it consists of all the transitions attached to a state that corresponds to both the start and end state. This means any sequence involving the connected transitions is permissible by the model. Though perfectly fitting with the log, such underfitting model does not convey much useful information to the user. In contrary, the process

model illustrated in Fig. 3 is much more precise than the flower model.

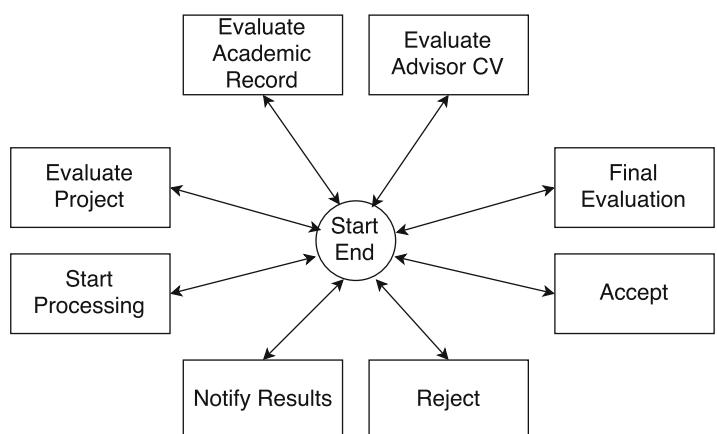
Generalization relates to a model's ability to account for yet to be observed behavior. Typically, an event log only represents a small fraction of the possible behavior in the process. As such, a good model must be generalizing enough so that unobserved but possible behavior is described. For example, if the model in Fig. 4 was discovered from an event log that contains only the trace *(Start Processing, Evaluate Project, Evaluate Academic Record, Evaluate Advisor CV, Final Evaluation, Accept, Notify Results)*, then the model would be both perfectly fitting and precise since all observed behaviors are captured by the model and that it does not allow any unseen behavior. Clearly, there is much unseen behavior that is very likely to occur in the future, e.g., the rejection of an application. This shows that, while it is important to have precise models, it is also important to avoid overfitting the observed behavior.

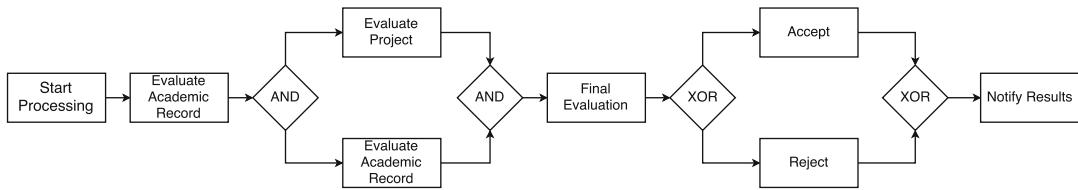
Some authors consider a fourth dimension called *simplicity*, relating to the model complexity, i.e., simple models should be preferred over



Conformance Checking, Fig. 1 Informal process model of a university scholarship process

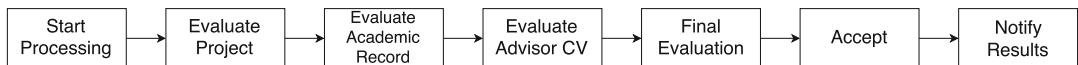
Conformance Checking,
Fig. 2 Imprecise flower model of the doctoral scholarship process



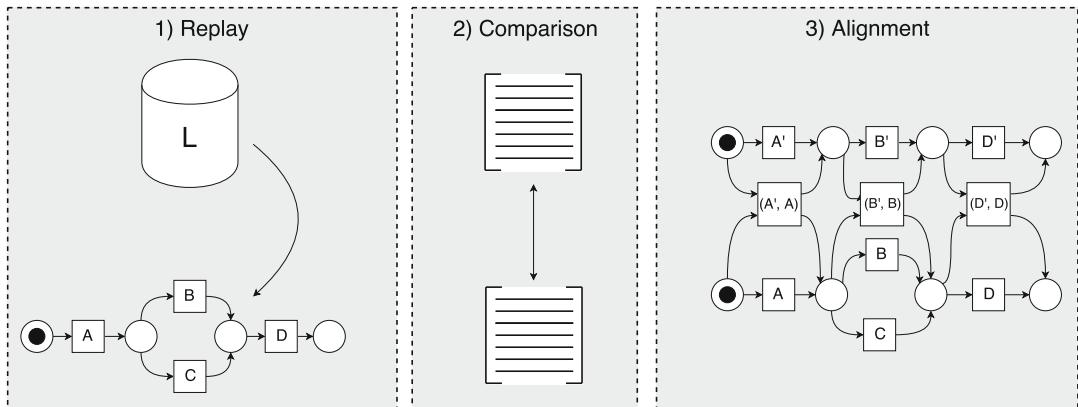


C

Conformance Checking, Fig. 3 Precise but unfitting model of the doctoral scholarship process



Conformance Checking, Fig. 4 Model that overfits one particular possible execution of the doctoral scholarship process



Conformance Checking, Fig. 5 Three main types of conformance checking approaches

complex models if both describe the same behavior. However this quality dimension relates only to the model and therefore is not normally measured by conformance checking techniques. This dimension is covered in the *Automated Process Discovery* entry of this encyclopedia.

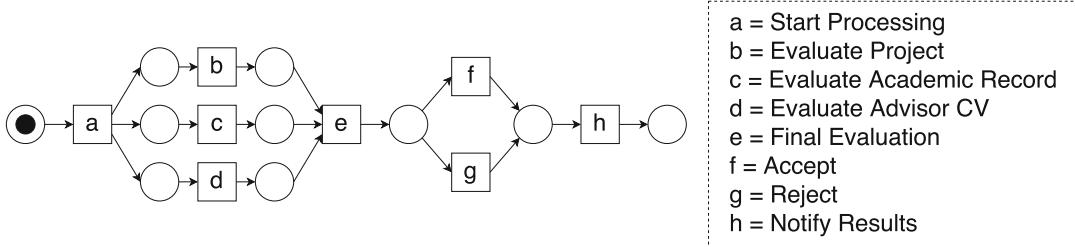
Overall, while the three quality dimensions are orthogonal to each other, in a real-life context, one is unlikely to find a pair of log and model that are in perfect conformance (i.e., perfectly fitting, precise, and generalizing). Often times, different scenarios may require different conformance levels and prioritization of the quality dimensions. For example, to analyze the well-established execution paths of a process, an analyst might prioritize fitness over the other dimensions. On the other hand, if an event log only contains a small number of cases, generalization would

likely to be prioritized over the other dimensions to account for possible future behavior.

Types of Conformance

Conformance checking techniques can be applied to understand and quantify these relationships between a log and model. There is a large collection of approaches and metrics that are based on different ways to compare a log and model.

Figure 5 shows that there are three main groups of conformance checking approaches – *replay*, *comparison*, and *alignment*. Replay-based approaches replay log traces onto the model and record information about the conformance during the replay. Process models can be denoted in different modeling notations, e.g., Business Process Modeling Notation (BPMN), Petri nets, and process trees, and each



Conformance Checking, Fig. 6 Model M_1 of the university doctoral scholarship process denoted in Petri net notation

representation bias has distinct characteristics, e.g., formalism and determinism. However, a proper process model is typically executable so that log traces can be re-executed stepwise by the model. Comparison-based approaches convert both the log and model into a common representation so that the log and model are directly comparable. Last but not least, alignment-based approaches seek to explain observed behavior in terms of modeled behavior by aligning log traces with the model. This brings conformance checking to the level of events and can offer detailed diagnosis on conformance issues.

Key Research Findings

In this section, two conformance checking approaches and three conformance metrics are presented.

Token Replay

Token replay is a replay-based conformance checking approach that measures the fitness between a log and model by replaying log traces onto process models denoted in the Petri net notation (Rozinat and van der Aalst 2008).

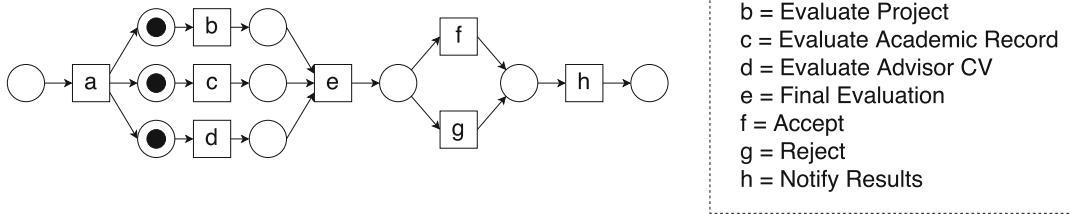
Consider model M_1 in Fig. 6 and log L_1 in Fig. 7. Model M_1 is denoted in Petri net notation so that the squares correspond to the activities in the process, filled circles correspond to tokens that mark the state of a process instance as activities get executed, and empty circles correspond to places that hold tokens. To execute an activity, all its input places (i.e., all the places connected by an incoming arrow to the activity) must have at least one token. This means that the activity

$$\begin{aligned}L_1 &= [t_1 = \langle a, b, c, d, e, f, h \rangle, \\&t_2 = \langle a, c, b, e, f, h \rangle, \\&t_3 = \langle a, b, d, c, g, e, h \rangle]\end{aligned}$$

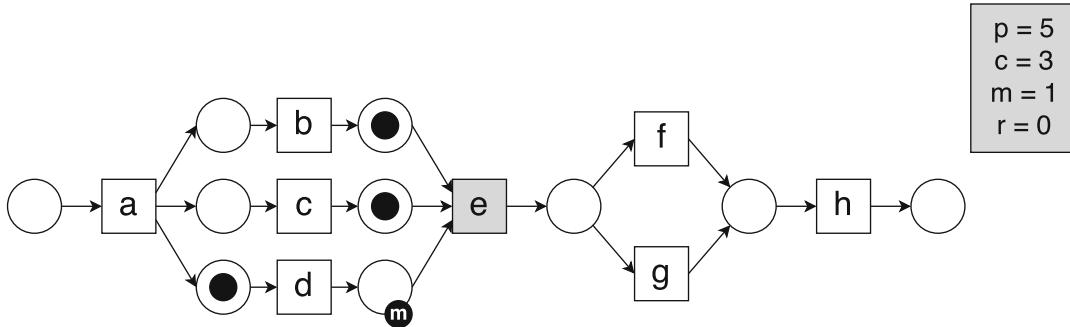
Conformance Checking, Fig. 7 Running example: event log L_1

is *enabled* and can be *fired*. When an activity is fired, the activity *consumes* a token from each of its input places before *producing* one token at each of its output places (i.e., all the places connected by an outgoing arrow from the activity). For example, activity a in model M_1 in Fig. 6 is currently enabled. If the activity is fired, it would consume the token of its input place and produce three tokens at each of its three output places as illustrated in Fig. 8. As such, an instance of the process can be recorded by successively firing enabled activities until no activities are enabled. For a valid Petri net model, an instance is initiated by having a token at each of the source places (i.e., places without any incoming arrows) and is deemed to be completed by firing activities until there is only one token at each of the sink places (i.e., places without any outgoing arrows) and none at any other places. The sequence of fired activities corresponds to a complete model trace, i.e., a possible execution of the model.

Log traces can be replayed onto the model by successively firing the activities related to each event in the log trace at an initiated Petri net model. If the log trace is perfectly fitting with the model, there should not be any problem with the replay since the log trace corresponds to a complete model trace. However, for deviating traces, replay would not be successful due to missing or



Conformance Checking, Fig. 8 Model M_1 after firing activity a



Conformance Checking, Fig. 9 Missing token to fire activity e in token replay of trace $t_2 = \langle a, c, b, e, f, h \rangle$

redundant tokens. An activity might be marked to be fired in the log trace but is not enabled in the model due to missing tokens at its input places. Consider the replay of trace t_2 in L_1 . Starting from the initial state of model M_1 as shown in Fig. 6, the firing of activity a, b, and c would consume three tokens and produce five tokens in the process. Figure 9 shows the state of model M_1 after the firing of the first three activities and records the number of consumed and produced tokens. According to trace t_2 , the next activity to be fired is activity e. However, this is not possible since one of the input places of activity e does not have any token, i.e., activity e is not enabled. To continue the replay, the missing token is artificially added into the empty input place, and the number of missing token is incremented. The rest of trace t_2 (activity f and g) can be replayed successively. Figure 10 shows that, after firing activity h, there is a remaining token in the input place of activity d since this activity was not fired in the replay of trace t_2 . As recalled, a process instance is only completed when there is only one token at each of the sink places and none at any other places. To complete the replay, the

remaining token is removed artificially, and the number of remaining tokens is incremented.

Based on the count of each token types consumed, produced, missing, and remaining ($p = 8$, $c = 8$, $m = 1$, $r = 1$), the fitness between model M_1 and trace t_2 can be computed as:

$$\begin{aligned} \text{fitness}(t_2, M) &= \frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p}) \\ &= \frac{1}{2}(1 - \frac{1}{5}) + \frac{1}{2}(1 - \frac{1}{5}) = 0.8 \end{aligned}$$

This fitness metric can be extended to the log level by considering the number of produced, consumed, missing, and remaining tokens from the token replay of all log traces.

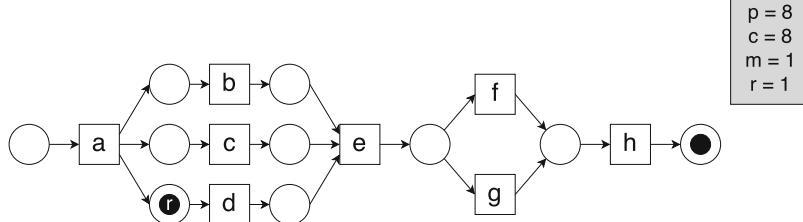
Cost-Based Alignment

The token replay approach can easily identify deviating traces in an event log. Moreover, the deviation severity can be compared using a fitness metric computed from the number of produced, consumed, missing, and remaining tokens. However, the token replay approach is prone to creat-

ing too many tokens for highly deviating traces so that any behavior is allowed. This can lead to an overestimation of the fitness. The approach is also specific to the Petri net notation. More importantly, in the case of a deviating trace, the approach does not provide a model explanation of the log trace. For example, the deviations in trace $t_2 = \langle a, c, b, e, f, h \rangle$ can be explained if it was considered with respect to the complete model trace $\langle a, c, b, d, e, f, h \rangle$. From this mapping, it is clear that the log trace is missing the execution of activity d (Evaluate Advisor CV). These mappings from log traces to model traces were introduced as *alignments* to address this limitation (van der Aalst et al. 2012).

Alignments are tables of two rows where the top row corresponds to the observed behavior (i.e., log projection) and the bottom row corresponds to the modeled behavior (i.e., model projection). Each column is therefore a move in the alignment where the observed behavior is aligned with the modeled behavior. Consider alignment γ_1 in Fig. 11. This alignment aligns

trace $t_3 = \langle a, b, d, c, g, e, h \rangle$ in L_1 and model M_1 in Fig. 6. The top row (ignoring \gg) yields the trace $t_3 = \langle a, b, d, c, g, e, h \rangle$, and the bottom row (ignoring \gg) yields a complete model trace $\langle a, b, d, c, e, g, h \rangle$. For each move in alignment γ_1 , the top row matches the bottom row if the step in the log trace matches the step in the model trace. This is called a *synchronous move*. In the case of deviations, a no-move symbol \gg is placed in the bottom row if there is a step in the log trace that cannot be mimicked by the model trace. For example, activity g is executed before activity e in trace t_3 , but model M_1 requires activity e to be fired before activity g. Hence, a *log move* is put where the top row has activity g and the bottom row has a no-move \gg . Similarly, a no-move symbol \gg is placed in the top row if there is a step in the model trace that cannot be mimicked by the log trace. For example, activity g is executed after activity e in the model trace according to model M_1 . Therefore, a *model move* is added where the top row has a no-move \gg and the bottom row has activity g. It is also possible



Conformance Checking, Fig. 10 Remaining token in token replay of trace $t_2 = \langle a, c, b, e, f, h \rangle$

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & b & d & c & g & e & \gg & h \\ \hline a & b & d & c & \gg & e & g & h \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & b & d & c & \gg & g & e & \gg & h \\ \hline a & b & \gg & c & d & \gg & e & g & h \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline a & b & d & c & \gg & g & e & h \\ \hline a & b & d & c & e & g & \gg & h \\ \hline \end{array}$$

Conformance Checking, Fig. 11 Possible alignments between trace $t_3 = \langle a, b, d, c, g, e, h \rangle$ in L_1 and model M_1 in Fig. 6

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline a & b & d & c & g & e & h & \gg \\ \hline \gg & a & b & d & c & e & g & h \\ \hline \end{array}$$

Conformance Checking, Fig. 12 Default alignment between trace t_3 in log L_1 and model M_1 in Fig. 6

that there are invisible transitions in the model which are not observable in the log. Similar to a model move, there would be a no-move in the top row and an invisible transition label in the bottom row. In total, there are four types of *legal moves* in an alignment: synchronous move, log move, model move, and invisible move.

For a particular log trace and model, there could be many possible alignments where each represents a different explanation of the observed behavior in terms of modeled behavior. For example, Fig. 11 shows three possible alignments between trace t_3 and model M_1 in Fig. 6. Clearly, alignment γ_1 and γ_3 are better alignments of trace t_3 and model M_1 than alignment γ_2 since they provide closer explanations with less log moves and model moves. The quality of an alignment can be quantified by assigning costs to moves. In general, model moves and log moves are assigned higher costs than synchronous moves because they represent deviations between modeled behavior and observed behavior. A standard cost assignment could be that all model moves and log moves are assigned a cost of 1 and synchronous moves and invisible moves are assigned a cost of 0. Invisible moves are normally assigned zero costs as they are related to invisible routing transitions in the model that are not observable in the log. Under the standard cost assignment, the costs of the alignments in Fig. 11 can be computed as follows:

$$\text{cost}(\gamma_1) = 0 + 0 + 0 + 0 + 1 + 0 + 1 + 0 = 2$$

$$\text{cost}(\gamma_2) = 0 + 0 + 1 + 0 + 1 + 1 + 0 + 1 + 0 = 4$$

$$\text{cost}(\gamma_3) = 0 + 0 + 0 + 0 + 1 + 0 + 1 + 0 = 2$$

This confirms the previous intuition that alignment γ_1 and γ_3 are better alignments than alignment γ_2 . Alignments with the minimal costs correspond to *optimal alignments* that give the clos-

est explanations of log traces in terms of modeled behavior. Note that there could be multiple optimal alignments for a particular log trace. For example, alignment γ_1 and γ_3 are both optimal alignments of trace t_3 under the standard cost assignment. Furthermore, optimal alignments are only optimal with respect to the given cost assignment. For example, alignment γ_1 would cease to be the optimal alignment if model moves and log moves of activity g are assigned a cost of 2 (i.e., $\text{cost}(\gamma_1) = 4$) to reflect that having deviations at the decision part of the process is quite severe. In practice, optimal alignments can be automatically found by finding the cheapest complete model trace in the synchronous product of the log trace and model using heuristic algorithms with proven optimality guarantees, e.g., the A* algorithm (van der Aalst et al. 2012).

Alignments can also be used to compute conformance metrics with respect to the different quality dimensions.

Cost-Based Fitness Metric

The fitness of a log trace and a model can be quantified by comparing the cost of an optimal alignment with the worst case scenario cost (Adriansyah 2014). In the worst scenario, the log trace is completely unfitting with the model. A default alignment between the two can be computed by assigning all the steps in the log trace as log moves and all the steps in the complete model trace as model moves. Since the optimal alignment minimizes the total alignment cost, the least costly complete model trace is used. Figure 12 shows the default alignment between trace t_3 and model M_1 under the standard cost assignment. The top row (ignoring \gg) yields trace t_3 , and the bottom row (ignoring \gg) yields a complete model trace $\langle a, b, d, c, e, g, h \rangle$. The cost-based fitness of trace t_3 can be computed as:

$$\begin{aligned} \text{fitness}(t_3, M) &= 1 - \frac{\text{cost}(\text{align}(t_3, M))}{\text{cost}(\text{align}_{\text{default}}(t_3, M))} \\ &= 1 - \frac{0 + 0 + 0 + 0 + 1 + 0 + 1 + 0}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1} \\ &= 1 - \frac{2}{14} = 0.857 \end{aligned}$$

where a fitness value of 1.0 means that the model and log trace are perfectly fitting.

Escaping Arc Precision

Precision based on escaping arcs can also be computed using alignments (Adriansyah et al. 2012). As previously mentioned, an imprecise model allows unobserved behavior, i.e., underfitting. For example, consider the Petri net model M_1 in Fig. 6 and the optimal alignments (under the standard cost assignment) between model M_1 and log L_1 in Fig. 13. Clearly, model M_1 is not perfectly precise as it allows for behavior that is not observed in log L_1 . According to model M_1 , activity b, c, and d can be executed in parallel following the execution of activity a. However, none of the log traces execute activity d after activity a. This imprecision in the model can be quantified by constructing a prefix automaton using the model projection of the alignments, i.e., the bottom row of the alignments. As previously presented, model projections of alignments explain potentially unfitting log traces in terms of modeled behavior so that they can be replayed on the process model. Figure 14 illustrates the constructed prefix automaton \mathcal{A}_1 for the alignments between log L_1 and model M_1 (ignoring the circles highlighted in red for now). Each prefix of the model projections of the alignments identifies a state (represented as circles), and the number in

the states corresponds to the weight. For example, the state $\langle a \rangle$ has a weight of 3 because it appears three times in the model projections (all three alignments start with activity a). On the other hand, the state $\langle a, c, b \rangle$ is only present in alignment γ_6 and therefore has a weight of 1. The states of automaton \mathcal{A}_1 represent states reached by the model during the execution of the log. For any particular state in automaton \mathcal{A}_1 , there might be activities that are enabled by the model but are not observed in the log execution. These activities indicate imprecisions of the model and are called *escaping arcs* of the model. Escaping arc states (represented as circles highlighted in red) are added to automaton \mathcal{A}_1 by replaying the automaton onto the model and checking for enabled activities at each state. For example, at state $\langle a \rangle$ (i.e., after firing activity a), activity b, c, and d are enabled as shown in Fig. 8. However, the prefix $\langle a, d \rangle$ was not observed in the construction of automaton \mathcal{A}_1 using log L_1 . This means that there is an escaping arc from state $\langle a \rangle$ to state $\langle a, d \rangle$, and this is added to the automaton by the state highlighted in red. The rest of the escaping arcs can be added in a similar way.

With the constructed prefix automaton, escaping arc precision can be computed by comparing the number of escaping arcs with the number of allowed arcs for all states:

$$\begin{aligned} precision(\mathcal{A}_1) &= 1 - \frac{\sum_{s \in S} \omega(s) \cdot |esc(s)|}{\sum_{s \in S} \omega(s) \cdot |mod(s)|} \\ &= 1 - \frac{3 \cdot 0 + 3 \cdot 1 + 2 \cdot 0 + \dots + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0}{3 \cdot 1 + 3 \cdot 3 + 2 \cdot 2 + \dots + 1 \cdot 2 + 1 \cdot 1 + 1 \cdot 1} \\ &= 1 - \frac{6}{36} = 0.833 \end{aligned}$$

where S is the set of states in automaton \mathcal{A}_1 , $\omega(\cdot)$ maps a state $s \in S$ to its weight, $esc(\cdot)$ maps a state $s \in S$ to its set of escaping arc states, and $mod(\cdot)$ maps a state $s \in S$ to its set of allowed states. A precision value of 1.0 indicates perfect precision, i.e., the model only allows observed behavior and nothing else.

Artificial Negative Events

Another approach to measure precision is through artificial negative events. Artificial negative events are induced by observing events that did not occur in the event log. These unobserved events (i.e., negative events) give information about things that are not allowed to

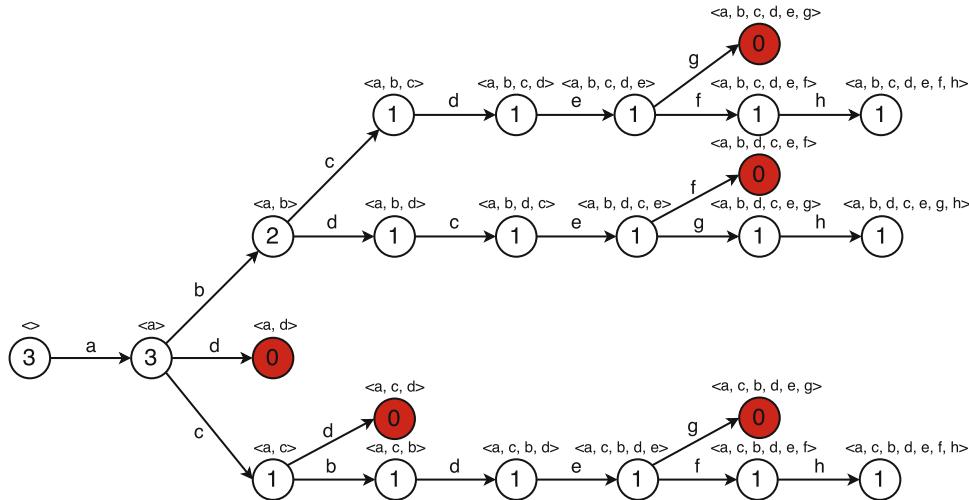
Conformance Checking,

Fig. 13 Optimal alignments between trace t_1, t_2, t_3 in $\log L_1$ and model M_1 in Fig. 6

$$\gamma_5 = \begin{vmatrix} a & b & c & d & e & f & h \\ a & b & c & d & e & f & h \end{vmatrix} \quad \gamma_6 = \begin{vmatrix} a & c & b & \gg & e & f & h \\ a & c & b & d & e & f & h \end{vmatrix}$$

$$\gamma_7 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & a & b & d & c & \gg & g & e & h \\ \hline a & b & d & c & e & g & \gg & h \\ \hline \end{array}$$

C



Conformance Checking, Fig. 14 Prefix automaton \mathcal{A}_1 of alignments between log L_1 and model M_1 enhanced with model behavior

occur in the process. Assuming that the event log gives a complete view of the process (i.e., a log completeness assumption), the precision of the process model can be computed using artificial negative events and the concepts of precision and recall in data mining.

Artificial negative events can be induced by grouping similar traces and then observing the events that did not occur for every event in the traces. Under the log completeness assumption, this means that these unobserved events are negative events that are not allowed to happen by the process (Goedertier et al. 2009).

The process model can then be compared with the log by treating the model as a predictive model. For a given incomplete event sequence (i.e., an unfinished process instance), activities that are permitted by the model and observed in the log are classified as true positives (TP). Activities that are permitted by the model but

are induced as negative events from the log are classified as false positive (FP). Activities that are not permitted by the model but observed in the log are classified as false negative (FN). Finally, activities that are not permitted by the model and are induced as negative events from the log are classified as true negative (TN). As shown in Fig. 15, precision and recall can be computed using a confusion matrix. Specifically, the precision of the positive class can be computed as:

$$precision = \frac{TP}{TP + FP}$$

The computed precision value corresponds to the precision of the three quality dimensions in process mining since it refers to the proportion of modeled behavior that is observed in the log.

The use of artificial negative events can also be extended to quantify generalization and to com-

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Conformance Checking, Fig. 15 Confusion matrix

pute a precision metric that is more robust against less complete event logs. This is achieved by extending the artificial negative event induction strategy to assign weights to the induced negative events (vanden Broucke et al. 2014).

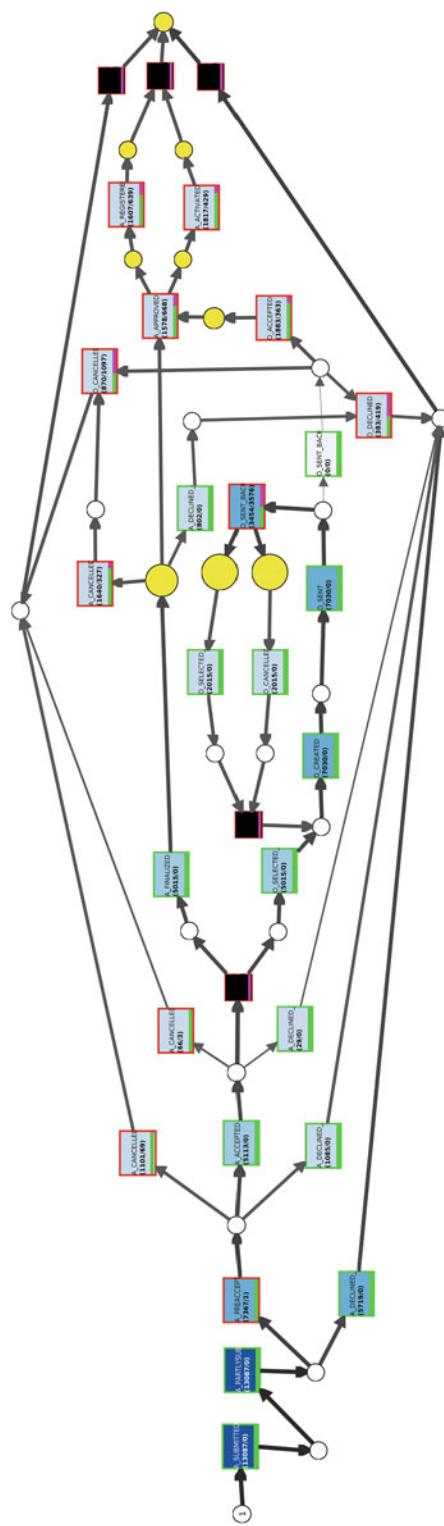
Examples of Application

All of the conformance checking techniques presented in the previous section have been implemented and are applicable to most real-life scenarios. In the following, the A* cost-based alignment technique is applied to a real-life dataset to illustrate how conformance checking can be applied to gain insights about a process. This example utilizes the data presented in the “Conformance Checking: What does your process do when you are not watching?” tutorial by de Leoni, van Dongen, and Munoz-Gama at the “15th International Conference on Business Process Management (BPM17).”

As previously presented, a process model and an event log are required to perform conformance checking. The real-life event log is taken from a Dutch financial institute and is of an application process for a personal loan or overdraft within a global financial organization (van Dongen 2012). This means that each case in the log records the occurred events of a particular loan application. The log contains some 262,200 events in

13,087 cases. Apart from some anonymization, the data is presented as it is recorded in the financial institute. The log is merged from three intertwined subprocesses so that the originating subprocess of each event can be identified by the first letter of the activity recorded by the event. In this example, the log is filtered so that it only contains events from two of the subprocesses: the process which records the state of the application (identifiable by “A_”) and the process which records the state of an offer communicated to the applicant (identifiable by “O_”). The model has been created with the help of domain experts and can be assumed to be a realistic representation of the underlying process.

Figure 16 shows the process model projected with the computed alignment results to allow a visual diagnosis of the conformance results. For each transition, there is an error bar to show the distribution of synchronous moves (green) and model moves (pink) for the transition. For example, there are 383 synchronous moves and 419 model moves related to transition *O_DECLINED*. The occurrence and amount of log moves are indicated by highlighting places in yellow and the size of the highlighted places. Observing the model, one can note that transition *O_SENT_BACK* is associated with a large amount of model moves. This transition is quite an important part of the process as it corresponds to the event where the financial institute receives a reply from the applicant after a loan offer is made. A model move of *O_SENT_BACK* in a log trace means that the system did not register a reply from the applicant regarding a made offer as required by the process for the corresponding loan application. Investigation of cases with a model move in *O_SENT_BACK* (e.g., the case with caseId 174036) would show that there are cases for which an offer was created, sent, and accepted without having received a reply from the corresponding applicant. Whether it was due to a system error, an employee’s mistake, or at worst a fraudulent case, clearly it is in the financial institute’s best interest to investigate the root cause of this conformance issue.



Conformance Checking, Fig. 16 Process model projected with alignment results

Future Directions for Research

While there have been significant advances in the research of conformance checking over the recent years, there are still many open challenges and research opportunities. Some of them include:

Conformance Dimensions

The proposed three quality dimensions (fitness, precision, and generalization) have been widely accepted, but there is still a need for further understanding on how to interpret and quantify them through metrics. Furthermore, conformance can be extended beyond the current three dimensions, e.g., log completeness to quantify whether if the observed event data gives the full picture of the underlying process (Janssenswillen et al. 2017).

Big Data and Real Time

Process mining techniques and tools are getting applied to larger and more complex processes. This means that they have to be scalable to handle the increased size and complexity. In fact, much of the recent research efforts in conformance checking have been focused on this issue. Related research lines include decomposed conformance checking (Munoz-Gama 2016) and online conformance checking for event streams (Burattin 2015).

Conformance Diagnosis and Process

Model Repair

It is not enough to just identify conformance issues; good diagnostic and visualization tools are crucial in helping the analyst identify and understand the root causes of the conformance issues. While there has been work done in this aspect of conformance checking, e.g., Buijs and Reijers (2014) and Munoz-Gama et al. (2014), there is much more to be done to provide better conformance diagnosis technology, e.g., new techniques and user study. Finally, once the differences between the model and the log have been diagnosed, the user may wish to repair the model in order to fix such differences and achieve a model that better describes the real process

executed. This topic is extensively covered in the *Process Model Repair* entry of this encyclopedia.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Business Process Event Logs and Visualization](#)
- ▶ [Process Model Repair](#)

References

- Adriansyah A (2014) Aligning observed and modeled behavior. Ph.D. thesis, Eindhoven University of Technology
- Adriansyah A, Munoz-Gama J, Carmona J, van Dongen BF, van der Aalst WMP (2012) Alignment based precision checking. In: Rosa ML, Soffer P (eds) Business process management workshops – BPM 2012 international workshops, Tallinn, Estonia, 3 Sept 2012. Revised papers. Lecture notes in business information processing, vol 132. Springer, pp 137–149
- Buijs, JCAM, Reijers HA (2014) Comparing business process variants using models and event logs. In: Enterprise, business-process and information systems modeling – 15th international conference, BPMDS 2014, 19th international conference, EMMSAD 2014, held at CAiSE 2014, Thessaloniki, Greece, 6–17 June 1 2014. Proceedings, pp 154–168
- Burattin A (2015) Process mining techniques in business environments – theoretical aspects, algorithms, techniques and open challenges in process mining. Lecture notes in business information processing, vol 207. Springer, Cham
- Goedertier S, Martens D, Vanthienen J, Baesens B (2009) Robust process discovery with artificial negative events. *J Mach Learn Res* 10:1305–1340
- Janssenswillen G, Donders N, Jouck T, Depaire B (2017) A comparative study of existing quality measures for process discovery. *Inf Syst* 71:1–15
- Munoz-Gama J (2016) Conformance checking and diagnosis in process mining – comparing observed and modeled processes. Lecture notes in business information processing, vol 270. Springer, Cham
- Munoz-Gama J, Carmona J, van der Aalst WMP (2014) Single-entry single-exit decomposed conformance checking. *Inf Syst* 46:102–122
- Rozinat A, van der Aalst WMP (2008) Conformance checking of processes based on monitoring real behavior. *Inf Syst* 33(1):64–95
- van der Aalst WMP (2013) Mediating between modeled and observed behavior: the quest for the “right” process: keynote. In: RCIS, pp 1–12. IEEE
- van der Aalst WMP (2016) Process mining – data science in action. Springer, Berlin/Heidelberg
- van der Aalst WMP, Adriansyah A, van Dongen BF (2012) Replay history on process models for

conformance checking and performance analysis. Wiley Interdisc Rev Data Min Knowl Disc 2(2):182–192
van Dongen BF (2012) BPI challenge 2012. 4TU Data-centrum. Eindhoven University of Technology
vanden Broucke SKLM, Weerdt JD, Vanthienen J, Baesens B (2014) Determining process model precision and generalization with weighted artificial negative events. IEEE Trans Knowl Data Eng 26(8):1877–1889

Consistency Criterion

- ▶ Database Consistency Models

Consistency Model

- ▶ Database Consistency Models

Consistency Models in MMOGs

- ▶ Transactions in Massively Multiplayer Online Games

Constraint-Based Process Mining

- ▶ Declarative Process Mining

Context-Aware User Mobility

- ▶ Using Big Spatial Data for Planning User Mobility

Continuous Dataflow Language

- ▶ Stream Processing Languages and Abstractions

Continuous Queries

Martin Hirzel

IBM Research AI, Yorktown Heights, NY, USA

C

Synonyms

Streaming SQL queries; Stream-relational queries; StreamSQL queries

Definitions

A continuous query in an SQL-like language is a declarative query on data streams expressed in a query language for streams derived from the SQL for databases.

Overview

Just like data that is stored in a relational database can be queried with SQL, data that travels in a stream can be queried with an SQL-like query language. For databases, the relational model and its language, SQL, have been successful because the relational model is a foundation for clean and rigorous mathematical semantics and because SQL is declarative, specifying what the desired result is without specifying how to compute it (Garcia-Molina et al. 2008). However, the classic relational model assumes that data resides in relations in a database. When data travels in a stream, such as for communications, sensors, automated trading, etc., there is a need for continuous queries. SQL dialects for continuous queries fill this need and inherit the advantages of SQL and the relational model. Furthermore, SQL-like streaming languages capitalize on the familiarity of SQL for developers and of implementation techniques from relational databases.

There are various different SQL-like streaming languages. This article illustrates concepts using CQL (the continuous query language, Arasu et al. (2006)) as a representative example, because it has clean semantics and addresses the interplay between streams and relations.

Section “[Findings](#)” explores other SQL-like streaming languages beyond CQL.

SQL-Like Syntax

The surface syntax for streaming SQL dialects borrows familiar SQL clauses (such as select,

from, and where) and augments them with streaming constructs (which turn streams into relations and vice versa). Consider the following CQL (Arasu et al. 2006) query with extensions adapted from Soulé et al. (2016):

```

1 Quotes : { ticker : string, ask : int } stream;
2 History : { ticker : string, low : int } relation;
3 Bargains : { ticker : string, ask : int, low : int } stream
4   = select istream(*)
5     from Quotes[now], History
6     where Quotes.ticker == History.ticker and Quotes.ask <= History.low;
```

Line 1 declares *Quotes* as a stream of {*ticker*, *ask*} tuples, and Line 2 declares *History* as a relation of {*ticker*, *low*} tuples. Neither *Quotes* nor *History* is defined with a query in the example. Lines 3–6 declare *Bargains* and define it with a query. Line 3 declares *Bargains* as a stream of {*ticker*, *ask*, *low*} tuples. Line 4 specifies the output using the **istream** operator, which creates a stream from the insertions to a relation, using * to pick up all available tuple attributes. Line 5 joins two relations, *Quotes[now]* and *History*, where *Quotes[now]* creates a relation from the

current contents of stream *Quotes*. Finally, Line 6 selects only tuples satisfying the given predicate, whereas any tuples for which the predicate is false are dropped.

The above example illustrates the core features of CQL, viz.: using operators such as **now** to turn streams into relations, using SQL to query relations, and using operators such as **istream** to turn relations into streams. For more detail, the following paragraphs explain the CQL grammar, starting with the top-level syntax:

```

program ::= decl+
decl   ::= ID `:` tupleType declKind (‘=’query)? `;`
tupleType ::= ‘{’(ID `:` TYPE)+, ‘}’
declKind ::= ‘relation’ | ‘stream’
query   ::= select from where? groupBy?
```

A program consists of one or more declarations. Each declaration has an identifier (*ID*), a tuple type (one or more attributes specified by their identifiers and types), a declaration kind (either **relation** or **stream**), and an optional query. The grammar meta-notation contains superscripts

for optional items (*X?*), repetition (*X⁺*), and repetition separated by commas (*X⁺,*). Finally, a query consists of mandatory select and from clauses and optional where and group-by clauses. Next, we look at the grammar for the select clause, which specifies the query output:

```

select      ::= ‘select’ outputList | ‘select’ relToStream ‘(’ outputList ‘)’
relToStream ::= ‘istream’ | ‘dstream’ | ‘rstream’
outputList  ::= ‘*’ | projectItem+, | aggrItem+,
projectItem ::= expr (‘as’ ID)?
aggrItem   ::= AGGR ‘(’ ID*, ‘)’(‘as’ ID)?
```

A select clause either specifies an output list directly or wraps an output list in a relation-to-stream operator. In the first case, the query output is a relation, while in the second case, the query output is a stream. There are three relation-to-stream operators: **istream** captures insertions, **dstream** captures deletions, and **rstream** captures the entire relation at any given point in time.

Here, a relation at a given point in time is a bag of tuples (i.e., an unordered collection of tuples that can contain duplicates). A stream is an unbounded bag of timestamped tuples (pairs of $\langle \text{timestamp}, \text{tuple} \rangle$). The grammar for *outputList* is borrowed from SQL. Next, we look at the grammar for the *from* clause, which specifies the query input:

```

from      ::= 'from' inputItem+,
inputItem ::= ID ('[' streamToRel ']')? ('as' ID)?
streamToRel ::= 'now' | 'unbounded' | timeWindow | countWindow
timeWindow ::= partitionBy? 'range' TIME ('slide' TIME)?
countWindow ::= partitionBy? 'rows' NAT ('slide' NAT)?
partitionBy ::= 'partition' 'by' ID+

```

An input item either identifies a relation directly or applies a stream-to-relation operator to a stream identifier. Stream-to-relation operators are written postfix in square brackets, reminiscent of indexing or slicing syntax in other programming languages. There are four such operators: **now** (tuples with the current timestamp), **unbounded** (tuples up to the current timestamp), **range** (time-based sliding window), and **rows** (count-based sliding window). Sliding windows can optionally be partitioned, in which case their size is determined separately and independently for each unique combination of the specified partitioning attributes. Sliding windows can optionally specify a slide granularity. Finally, we look at the grammar for *where* and *groupBy* as examples of other classic SQL clauses:

```

where    ::= 'where' expr
groupBy ::= 'group' 'by' ID+

```

The *where* clause selects tuples using a predicate expression, and the *group-by* clause specifies the scope for aggregation queries. These clauses in CQL are borrowed unchanged from SQL. For brevity, we omitted other classic SQL constructs, such as *distinct*, *union*, *having*, or other types of joins, which streaming SQL dialects such as CQL can borrow from SQL as is.

Typically, before execution, queries in SQL or its derivatives are first translated into a logical query plan of operators, which is the subject of the next section.

Stream-Relational Algebra

Whereas the SQL-like syntax of the previous section is designed to be authored by humans, this section describes an algebra designed to be optimized and executed by stream-query engines. The algebra is stream-relational because it augments with stream operators the relational algebra from databases. Relational algebra has well-understood semantics (Garcia-Molina et al. 2008), and the CQL authors rigorously defined the formal semantics for the additional streaming operators (Arasu and Widom 2004). The following paragraphs provide only an informal overview of the operators; interested readers can consult the literature for formal treatments. The notation for operator signatures is:

$$\text{operator}_{\langle \text{configuration} \rangle}(\text{input}) \rightarrow \text{output}$$

The configuration of an operator specializes its behavior. The input to an operator consists of one or multiple relations or a stream. And the output of an operator consists of either a relation or a stream. Relational algebra is compositional, since the output of one operator can be plugged

into the input of another operator, modulo compatible kind and type. For instance, the stream-relational algebra for stream *Bargains* in the CQL

$$\text{istream}\left(\sigma_{\langle \text{ask} \leq \text{low} \rangle} \left(\bowtie_{\langle \text{Quotes.ticker} = \text{History.ticker} \rangle} (\text{now}(\text{Quotes}), \text{History}) \right) \right)$$

Classic relational algebra has operators from relations to relations (Garcia-Molina et al. 2008):

- $\pi_{\langle \text{assignments} \rangle}(\text{relation}) \rightarrow \text{relation}$
Project each input tuple using assignments to create a tuple in the output relation.
- $\sigma_{\langle \text{predicate} \rangle}(\text{relation}) \rightarrow \text{relation}$
Select tuples for which the predicate is true, and filter out tuples for which it is false.
- $\bowtie_{\langle \text{predicate} \rangle}(\text{relation}^+) \rightarrow \text{relation}$
Join tuples from input relations as if with a cross product followed by $\sigma_{\langle \text{predicate} \rangle}$.
- $\gamma_{\langle \text{groupBy}, \text{assignments} \rangle}(\text{relation}) \rightarrow \text{relation}$
Group tuples and then aggregate within each group, using the given assignments.

For brevity, we omitted other classic relational operators, but they could be added trivially, thanks to the compositionality of the algebra. CQL can use the well-defined semantics of classic relational algebra operators by applying them on snapshots of relations at a point in time. Some operators, such as π and σ , process each tuple in isolation, without carrying any state from one tuple to another (Xu et al. 2013). These operators could be easily lifted to work on streams, and indeed, some streaming SQL dialects do just that. But the same is not true for stateful operators such as \bowtie and γ . To use these on streams, CQL first converts streams to relations, using window operators:

- **now(stream) \rightarrow relation**
At each time instant t , all tuples from the input stream with timestamp exactly t .
- **unbounded(stream) \rightarrow relation**
At each time instant t , all tuples from the input stream with timestamp at most t .

example from the start of section “[SQL-Like Syntax](#)” is:

- **range_{(partitionBy, size, slide)?}(stream) \rightarrow relation**
Use a time-based sliding window on the input stream as the output relation.
- **rows_{(partitionBy, size, slide)?}(stream) \rightarrow relation**
Use a count-based sliding window on the input stream as the output relation.

These window operators correspond directly to the corresponding surface syntax discussed in section “[SQL-Like Syntax](#)”. Gedik surveyed these and more window constructs and their implementation (Gedik 2013). A final set of operators turns relations back into streams:

- **istream(relation) \rightarrow stream**
Watch input relation for insertions, and send those as tuples on the output stream.
- **dstream(relation) \rightarrow stream**
Watch input relation for deletions, and send those as tuples on the output stream.
- **rstream(relation) \rightarrow stream**
At each time instant, send all tuples currently in input relation on output stream.

This article illustrated continuous queries in SQL-like languages using CQL as the concrete example, because it is clean and well studied. The original CQL work contains more details and a denotational semantics (Arasu et al. 2006; Arasu and Widom 2004). Soulé et al. furnish CQL with a static type system and formalize translations from CQL via stream-relational algebra to a calculus with an operational semantics (Soulé et al. 2016).

Findings

CQL was not the first dialect of SQL for streaming. TelegraphCQ reused the front-end of PostgreSQL as the basis for its surface language (Chandrasekaran et al. 2003). Rather than focusing on surface language innovation, TelegraphCQ focused on a stream-relational algebra back-end that pioneered new techniques for dynamic optimization and query sharing. Gigascope had its own dialect of SQL called GSQL (Cranor et al. 2003). Unlike CQL, GSQL used an algebra where all operators work directly on streams. As discussed earlier, this is straightforward for π and σ , but not for \bowtie and γ . Therefore, GSQL required \bowtie and γ to be configured with constraints over ordering attributes that effectively function as windows. Aurora used a graphical interface for surface-level programming, but we still consider it an SQL-like system, because it used a stream-relational algebra (Abadi et al. 2003). Aurora’s Stream Query Algebra (SQuAl) contained the usual operators π , σ , \bowtie , and γ , as well as union, sort, and a resample operator that interpolates missing values.

CQL took a more language-centric approach than its predecessors. It also inspired work probing semantic subtleties in SQL-like streaming languages. Jain et al. precisely define the semantics for the corner case of StreamSQL behavior where multiple tuples have the same timestamp (Jain et al. 2008). In that case, there is no inherent order among these tuples, so tuple-based windows must choose arbitrarily, leading to undefined results. Furthermore, if actions are triggered on a per-tuple basis, there can be multiple actions at a single timestamp, leading to spurious intermediate results that some would consider a glitch. The SECRET paper is also concerned with problems of time-based sliding windows (Botan et al. 2010). SECRET stands for ScopE (which timestamps belong to a window), Contents (which tuples belong to a window), REport (when to output results), and Tick (when to trigger computation). Finally, Zou et al. explored turning repeated SQL queries into con-

tinuous CQL queries by turning parameters that change between successive invocations into an input stream (Zou et al. 2010).

Today, there is still much active research on big-data streaming systems, but the focus has shifted from SQL dialects to embedded domain-specific languages (EDSLs, Hudak 1998). An EDSL for streaming is a library in a host language that offers abstractions for continuous queries. In practice, most EDSLs lack the rigorous semantics of stand-alone languages such as CQL but have the advantage of posing a lower barrier to entry for developers who are already proficient in the host language, being easier to extend with user-defined operators, and not requiring a separate language tool-chain (compiler, debugger, integrated development environment, etc.).

C

Examples

The beginnings of sections “[SQL-Like Syntax](#)” and “[Stream-Relational Algebra](#)” show a concrete example of the same query in CQL and in stream-relational algebra, respectively. The most famous example of a set of continuous queries written in an SQL-like language is the Linear Road benchmark. The benchmark consists of computing variable-rate tolling for congestion pricing on highways. A simplified version of Linear Road serves to motivate and illustrate CQL (Arasu et al. 2006). The full version of Linear Road is presented in a paper of its own (Arasu et al. 2004). Both the simplified version and the full version of the benchmark continue to be popular, and not just for SQL-inspired streaming systems and languages (Grover et al. 2016; Hirzel et al. 2016; Jain et al. 2006; Soulé et al. 2016).

Future Directions for Research

Stream processing is an active area of research, and new papers often use a streaming SQL foundation to present their results. One challenging issue for streaming systems is how to handle

out-of-order data. For instance, CEDR suggests a solution based on stream-relational algebra using several timestamps per tuple (Barga et al. 2007). One challenge that is particular to SQL-like languages is how to extend them with user-defined operators without losing the well-definedness of the restricted algebra. For instance, StreamInsight addresses this issue with an extensibility framework (Ali et al. 2011). Finally, the semantics of SQL are defined by reevaluating relational operators on windows whenever the window contents change. Nobody suggests that this reevaluation is the most efficient approach, but developing better solutions is an interesting research challenge. For instance, the DABA algorithm performs associative sliding-window aggregation on FIFO windows in worst-case constant time (Tangwongsan et al. 2017). All three of the abovementioned research topics (out-of-order processing, extensibility, and incremental streaming algorithms) are still ripe with open issues.

Cross-References

► Stream Processing Languages and Abstractions

References

- Abadi DJ, Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. *J Very Large Data Bases (VLDB J)* 12(2):120–139
- Ali M, Chandramouli B, Goldstein J, Schindlauer R (2011) The extensibility framework in Microsoft StreamInsight. In: International conference on data engineering (ICDE), pp 1242–1253
- Arasu A, Widom J (2004) A denotational semantics for continuous queries over streams and relations. *SIGMOD Rec* 33(3):6
- Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbets R (2004) Linear road: a stream data management benchmark. In: Conference on very large data bases (VLDB), pp 480–491
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. *J Very Large Data Bases (VLDB J)* 15(2):121–142
- Barga RS, Goldstein J, Ali M, Hong M (2007) Consistent streaming through time: a vision for event stream processing. In: Conference on innovative data systems research (CIDR), pp 363–373
- Botan I, Derakhshan R, Dindar N, Haas L, Miller RJ, Tatbul N (2010) SECRET: a model for analysis of the execution semantics of stream processing systems. In: Conference on very large data bases (VLDB), pp 232–243
- Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein JM, Hong W, Krishnamurthy S, Madden S, Raman V, Reiss F, Shah MA (2003) TelegraphCQ: continuous dataflow processing for an uncertain world. In: Conference on innovative data systems research (CIDR)
- Cranor C, Johnson T, Spataschek O, Shkapenyuk V (2003) Gigascope: a stream database for network applications. In: International conference on management of data (SIGMOD) industrial track, pp 647–651
- Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Pearson/Prentice Hall, London, UK
- Gedik B (2013) Generic windowing support for extensible stream processing systems. *Softw Pract Exp (SP&E)* 44:1105–1128
- Grover M, Rea R, Spicer M (2016) Walmart & IBM revisit the linear road benchmark. <https://www.slideshare.net/RedisLabs/walmart-ibm-revisit-the-linear-road-benchmark> (Retrieved Feb 2018)
- Hirzel M, Rabiah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60
- Hudak P (1998) Modular domain specific languages and tools. In: International conference on software reuse (ICSR), pp 134–142
- Jain N, Amini L, Andrade H, King R, Park Y, Selo P, Venkatramani C (2006) Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In: International conference on management of data (SIGMOD), pp 431–442
- Jain N, Mishra S, Srinivasan A, Gehrke J, Widom J, Balakrishnan H, Cetintemel U, Cherniack M, Tibbets R, Zdonik S (2008) Towards a streaming SQL standard. In: Conference on very large data bases (VLDB), pp 1379–1390
- Soulé R, Hirzel M, Gedik B, Grimm R (2016) River: an intermediate language for stream processing. *Softw Pract Exp (SP&E)* 46(7):891–929
- Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Conference on distributed event-based systems (DEBS), pp 66–77
- Xu Z, Hirzel M, Rothermel G, Wu KL (2013) Testing properties of dataflow program operators. In: Conference on automated software engineering (ASE), pp 103–113
- Zou Q, Wang H, Soulé R, Hirzel M, Andrade H, Gedik B, Wu KL (2010) From a stream of relational queries to distributed stream processing. In: Conference on very large data bases (VLDB) industrial track, pp 1394–1405

Continuous Query Optimization

► Stream Query Optimization

Coordination Avoidance

Faisal Nawab

University of California, Santa Cruz, CA, USA

Definitions

Coordination avoidance denotes a class of distributed system methods that minimize the amount of coordination between nodes while maintaining the integrity of the application.

Overview

In many data management systems, data and processing are replicated or distributed across nodes (Kemme et al. 2010; Bernstein and Goodman 1981). This replication and distribution increase the levels of fault tolerance and availability. However, they introduce a coordination cost to maintain the integrity of applications. Since nodes are processing data for the same application simultaneously, there is the possibility of conflicting operations that may overwrite the work of other nodes. To overcome this problem, coordination and synchronization protocols have been developed to ensure the integrity of data. Typically, the coordination protocols strive to ensure a guarantee of correctness, such as serializability (Bernstein et al. 1987) and linearizability (Herlihy and Wing 1990). These are strong notions of correctness that model the correctness of a distributed system by whether the data management system can mask the underlying concurrency from the upper layers. Serializability, for example, guarantees that the outcome of the execution of distributed transactions (i.e., bundles of operations) is equivalent to some serial execution.

Thus, the application developer has the illusion of the existence of a single machine that executes transactions serially.

A strong guarantee of correctness is extremely useful for application developers because it frees them from thinking about concurrency and replication issues. Additionally, relying on the application level to deal with concurrency anomalies is error-prone and may lead to many wasted efforts in applications reinventing the wheel. Rather, the data management approach separates concerns and relies on a data management layer to manage concurrency and replication to guarantee the consistency of the application. However, these approaches are expensive, requiring extensive coordination between nodes, as any two operations touching the same data item need to be synchronized. Data management systems tackled this problem by building better coordination and synchronization designs and protocols to optimize performance. Unfortunately, this continues to be a challenging task as distributed systems are becoming bigger with large-scale many-node and many-core deployments that increase the coordination and communication demands. Also, distributed systems are increasingly being deployed across wide geographical locations, increasing the latency of communication. These two trends continue to be adopted and are made accessible through cloud technology. To reduce the cost of coordination, there have been approaches that relax the consistency guarantees and trade them off with better performance. This approach, however, does not retain the easy-to-use data management abstractions that can be provided with strong consistency guarantees.

Coordination avoidance aims to reduce the cost of coordination while maintaining the integrity of data and the easy-to-use data management abstractions. The approach of coordination avoidance exploits knowledge of the application layer to extract the set of consistency guarantees that are sufficient for its application-level correctness. Therefore, the cost of coordination is only incurred when there is a potential that it will lead to an anomaly. Extensions to coordination avoidance approaches build execution

models and abstractions that facilitate and enable avoiding coordination.

Key Research Findings

The goal of coordination avoidance techniques and protocols is to minimize the cost of coordination while retaining the application integrity. There have been a plethora of work that investigates such approaches. What these works have in common is attempting to leverage application-level semantics to deduce the set of guarantees that are sufficient for a consistent execution. They may take various forms that are presented in this section. Coordination avoidance techniques combine a subset of different approaches. Generally, a coordination avoidance protocol tackle two tasks: extracting or modeling a set of consistency semantics for a particular application and executing operations while minimizing coordination to the cases when only the set of extracted consistency semantics are threatened.

There are various approaches for extracting consistency semantics that vary in their ease of use, generality, and optimality, where coordination avoidance protocols typically face the trade-off between these three properties. One approach extracts consistency semantics from existing code via the use of static analysis or other similar methods. This approach requires the least intervention from application developers. However, it introduces the challenge of extracting features from general code which can be limited in its scope. Some other approaches rely on the application developers to annotate or model their code to enable extracting the consistency semantics. This approach requires additional intervention from the application developer, although allowing the use of existing general code. Alternatively, some works explore an approach where specialized frameworks and abstractions are introduced to facilitate and enable the extraction of consistency semantics. With the extracted features, there are various approaches to execute distributed applications that also vary in their ease of use, generality, and optimality. One approach is to selectively coordinate between nodes according

to the extracted consistency semantics. Another approach is to provide specialized protocols and execution mechanisms that guarantee the consistency semantics. The following are examples of coordination avoidance methods and work that combine and explore different extraction and execution approaches.

Commutativity and Convergence

A large number of coordination avoidance protocols leverage the commutativity of operations. Commutative operations are ones that can be reordered arbitrarily while reaching the same final state. Therefore, commutative operations can execute at different nodes in different orders without synchronously coordinating with other nodes. The operations are asynchronously propagated to other nodes that execute them without the need of any ordering. Commutative operations are especially useful for hot spots that are frequently being updated, such as counters. Many attempts have been made to generalize the notion of commutativity in the context of data management (Badrinath and Ramamritham 1992; Korth 1983). Recoverability (Badrinath and Ramamritham 1992) generalizes commutativity to include more operations by explicitly ordering commit points. For example, two operations pushing to a stack simultaneously are not commutative. However, with recoverability, they can execute concurrently while maintaining serializability with the condition that they commit in the order they are invoked. Korth (1983) generalizes locking to account for the existence of commutative operations and thus allow more concurrency when commutative operations are being used. Commutative operations can also be used in conjunction with other consistency guarantees. Walter (Sovran et al. 2011), for example, is a system that employs a consistency notion based on snapshot isolation. Walter allows commutative operations to be performed without coordination.

Convergent and commutative replicated data types (CRDTs) (Shapiro et al. 2011) are an approach to make the use of commutativity more powerful by designing access abstractions and data structures that are more general than

traditional commutative operations while still maintaining commutativity. This has led to the development of nontrivial data structures that perform complex operations while being commutative, thus requiring no coordination (Preguica et al. 2009). CRDTs are also used in the context of eventual consistency by providing a means to converge operations to the same, eventually consistent state. Also, it is used in coordination-free protocols to converge operations that do not need to be synchronously coordinated (Bailis et al. 2014).

CALM (Alvaro et al. 2011) is a principle that shows that a distributed program can be executed without coordination if the program's logic is monotone. This enables avoiding coordination by building programs that ensure logical monotonicity in addition to using analysis techniques to assist whether a given program is logically monotonic. Bloom is a domain-specific declarative language that facilitates leveraging the CALM principle to build coordination-free programs. In this approach, the whole program does not have to be monotonic, rather, it is possible to detect potential anomalies that need to be addressed by the developer.

Application-Level Correctness Semantics

Many data management system works utilize application-level invariants and correctness semantics to avoid coordination (Lamport 1976; Agrawal et al. 1993, 1994; Garcia-Molina 1983; Li et al. 2012; Roy et al. 2015; Bailis et al. 2014). In this approach, operations are allowed to execute and commit without coordination if the application-level correctness semantics are not violated. Lamport (1976) (as described in Agrawal et al. 1993) is the first to show that application-level semantics can be used to avoid coordination between certain types of transactions. Garcia-Molina (1983) introduced a design of a concurrency control protocol that leverages application-level correctness semantics. The protocol receives application-level semantics from users and then uses them to allow nonserializable schedules that do not violate the application's correctness semantics. To achieve this, the protocol divides

the transaction into steps and derives what steps can interleave with each other. Other works adopt a similar approach of dividing the transaction to find interleaving between the parts of the transaction. A set of work has shown that performing this division hierarchically may lead better results (Lynch 1983; Weikum 1985; Farrag and Özsu 1989). A formalization of the correctness of application-level consistency definitions is provided in Korth and Speegle (1988).

Defining how the application-level consistency semantics are expressed is important for the practicality of the solution (i.e., how accessible is the interface to developers) and the performance gains of the solution (i.e., how the interface can maximize the amount of captured knowledge). Agrawal et al. (1993) propose the use of special operations, called consistency assertions, that captures a user's view of the database and the criterion for correct concurrency execution. Two correctness criteria are developed to leverage the special operations by making the database allow concurrent execution of two transactions if the consistency assertions are not violated. Relative serializability (Agrawal et al. 1994) extends the transaction model by allowing users to define the atomicity requirements of transactions relative to each other. Then, it is possible to interleave parts of transactions that do not violate the defined atomicity requirements.

The use of application-level correctness invariants is especially important for large-scale and geo-scale deployments where the coordination cost is high. This has motivated the development of more solutions that use application-level correctness invariants for these new architectures (Li et al. 2012; Roy et al. 2015; Bailis et al. 2014; Zhang et al. 2013). Gemini (Li et al. 2012) enables the coexistence of fast, eventually consistent operations with strongly consistent, slow operations. Therefore, operations are executed with high performance, and the cost of enforcing consistency is only paid when necessary. To facilitate this, a consistency model, called RedBlue consistency, is proposed. Using the consistency formulation and application-level consistency invariants, operations are assigned to

be either fast or consistent. The assignment is made with the guarantee that executing fast operations without synchronous coordination would not lead to violations of the application-level consistency semantics. Specifically, fast operations execute locally at one of the data centers and then lazily propagate to other data centers. Fast operations are only guaranteed to be causally ordered. Consistent operations, on the other hand, are synchronously synchronized, potentially with other data centers, leading to high latency. To be either the Homeostasis protocol (Roy et al. 2015) aims at automating the process of generating application-level correctness invariants. A program analysis technique is used to extract the application-level consistency invariants from code. In the Homeostasis protocol, the target consistency level is serializability, and the code is analyzed to extract the invariants that will lead to a serializable execution as observed by users. Application developers can then refine the extracted application-level correctness invariants. The Homeostasis protocol may enable and inspire future work on the use of automatic extraction of correctness invariants that improve their accuracy. However, it is also useful as a tool to bootstrap the process of specifying application-level correctness invariants.

Transaction chains (Zhang et al. 2013) leverage transaction chopping (Shasha et al. 1995) to commit transactions in a geo-scale deployment locally without waiting for other data centers. Transaction chopping (Shasha et al. 1995) is a theoretical framework that enables decomposing transactions to smaller pieces. The original goal of transaction chopping is to allow more concurrency. Transaction chains (Zhang et al. 2013) use the concepts of transaction chopping to decompose transaction. Then, it introduces constraints on the execution of transactions. Transaction parts are executed in the same order at the different nodes (potentially at different data centers). Also, instances of the same transaction are ordered at the first node that handles that type of transactions. It turns out that these two constraints, in addition to transaction chopping, allow guaranteeing the success of the execution of the transaction after the success in the first

node. Thus, rather than waiting for the response from multiple nodes at multiple data centers, transaction can execute and be guaranteed to succeed after execution at the first node only. Transaction chains incorporate many annotation and commutative techniques to enable more flexibility in the schedules and constraints that are enforced on transactions.

Weak and Relaxed Consistency

Another method to avoid coordination is to weaken or relax the consistency guarantees. Eventual consistency (Cooper et al. 2008; DeCandia et al. 2007; Bailis and Ghodsi 2013) only guarantees that the data copies are eventually consistent in the absence of updates. Therefore, coordination can be performed asynchronously, and convergence can be achieved via commutative and converging operators. Causality (Lamport 1978) is a stronger guarantee that ensures that events at one node are totally ordered and events at one node are ordered after everything that have been received by the point they are generated. Causality does not require synchronous coordination, which made it a candidate for many environments with high coordination costs (Lloyd et al. 2011, 2013; Bailis et al. 2013; Nawab et al. 2015; Du et al. 2013). Extensions of causality that consider causal transactions rather than operations or events are also shown to require no synchronous coordination (Lloyd et al. 2011, 2013; Nawab et al. 2015). Parallel snapshot isolation (PSI) (Sovran et al. 2011) extends snapshot isolation (Berenson et al. 1995) for geo-scale environments. This extension allows many types of transactions to execute without the need for synchronous coordination between data centers. Transactions with write-write conflicts, however, are still required to synchronously coordinate.

Versioning and Snapshots

Serializability is a guarantee that the outcome of transactions mimics an outcome of a serial execution. This makes read-only transactions have the opportunity to execute without coordination by reading from a consistent snapshot of data.

The outcome of such read-only transaction might be stale but consistent. However, the challenge in read-only transactions is in the development of methods that minimize the interference with read-write transactions. This is especially the case for analytics queries that translate to large, long-lived read-only transactions. Schlageter (1981) showed that read-only transactions can be executed without interfering with read-write transactions. Specifically, in an optimistic concurrency control protocol, read-only transactions do not need to be validated as long as they reflect the consistent state of the database. However, read-write transactions wait for read-only transactions to finish. This may lead to long delays with large read-only transactions. To overcome this limitation, multi-versioning techniques were proposed to execute read-only transactions on a recent, possibly stale snapshot of the database (Agrawal et al. 1987). In a replicated system, it was shown that a read-only transaction can avoid coordination with other replicas by reading a local, consistent copy (Satyanarayanan and Agrawal 1993). This entails modifying the coordination of read-write transactions and additional overhead to enable creating a consistent snapshot locally at every replica. Recently, Spanner (Corbett et al. 2012) shows the use of accurate time synchronization to serve consistent read-only transactions with improved freshness.

Conclusion

Coordination is continuing to be a major challenge in data management systems due to large-scale and geo-scale deployments. Coordination avoidance techniques enable reducing the overhead of coordination, thus improving performance, while maintaining the ease of use and intuition of data management abstractions and strong consistency guarantees. There have been many ways that can be used individually or together to avoid coordination such as leveraging commutativity and convergence, application-level consistency semantics, relaxed consistency guarantees, and specialized transaction types.

These solutions provide different levels of ease of use, generality, and optimality, making them suitable for different classes of applications and use cases.

C

Examples of Application

Coordination avoidance is used in traditional and cloud data management systems.

Future Directions for Research

The emergence of many new technologies, system architectures, and applications invites the investigation of new coordination avoidance techniques and adaptations of existing coordination avoidance technique.

Cross-References

- ▶ [Geo-Replication Models](#)
- ▶ [Geo-Scale Transaction Processing](#)
- ▶ [Weaker Consistency Models/Eventual Consistency](#)

References

- Agrawal D, Bernstein AJ, Gupta P, Sengupta S (1987) Distributed optimistic concurrency control with reduced rollback. *Distrib Comput* 2(1):45–59. <https://doi.org/10.1007/BF01786254>
- Agrawal D, El Abbadi A, Singh AK (1993) Consistency and orderability: semantics-based correctness criteria for databases. *ACM Trans Database Syst (TODS)* 18(3):460–486
- Agrawal D, Bruno JL, El Abbadi A, Krishnaswamy V (1994) Relative serializability (extended abstract): an approach for relaxing the atomicity of transactions. In: Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. ACM, pp 139–149
- Alvaro P, Conway N, Hellerstein JM, Marczak WR (2011) Consistency analysis in bloom: a calm and collected approach. In: CIDR, pp 249–260
- Badrinath B, Ramamirtham K (1992) Semantics-based concurrency control: beyond commutativity. *ACM Trans Database Syst (TODS)* 17(1):163–199
- Bailis P, Ghodsi A (2013) Eventual consistency today: limitations, extensions, and beyond. Queue

- 11(3):20:20–20:32. <https://doi.org/10.1145/2460276.2462076>
- Bailis P, Ghodsi A, Hellerstein JM, Stoica I (2013) Bolt-on causal consistency. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD'13. ACM, New York, pp 761–772. <https://doi.org/10.1145/2463676.2465279>
- Bailis P, Fekete A, Franklin MJ, Ghodsi A, Hellerstein JM, Stoica I (2014) Coordination avoidance in database systems. Proc VLDB Endow 8(3):185–196
- Berenson H, Bernstein P, Gray J, Melton J, O’Neil E, O’Neil P (1995) A critique of ansi SQL isolation levels. ACM SIGMOD Rec 24:1–10. <https://doi.org/10.1145/223784.223785>
- Bernstein PA, Goodman N (1981) Concurrency control in distributed database systems. ACM Comput Surv (CSUR) 13(2):185–221
- Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Reading
- Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R (2008) Pnutes: Yahoo!’s hosted data serving platform. Proc VLDB Endow 1(2):1277–1288. <https://doi.org/10.14778/1454159.1454167>
- Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D (2012) Spanner: Google’s globally-distributed database, pp 251–264. <http://dl.acm.org/citation.cfm?id=2387880.2387905>
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon’s highly available key-value store. In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP’07. ACM, New York, pp 205–220. <https://doi.org/10.1145/1294261.1294281>
- Du J, Elnikety S, Roy A, Zwaenepoel W (2013) Orbe: scalable causal consistency using dependency matrices and physical clocks. In: Proceedings of the 4th annual symposium on cloud computing, SOCC’13. ACM, New York, pp 11:1–11:14. <https://doi.org/10.1145/2523616.2523628>
- Farrag AA, Özsu MT (1989) Using semantic knowledge of transactions to increase concurrency. ACM Trans Database Syst (TODS) 14(4):503–525
- Garcia-Molina H (1983) Using semantic knowledge for transaction processing in a distributed database. ACM Trans Database Syst (TODS) 8(2):186–213
- Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492. <https://doi.org/10.1145/78969.78972>
- Kemme B, Jimenez-Peris R, Patino-Martinez M (2010) Database replication. Synth Lect Data Manag 2(1):1–153. <http://www.morganclaypool.com/doi/abs/10.2200/S00296ED1V01Y201008DTM007>
- Korth HF (1983) Locking primitives in a database system. J ACM 30(1):55–79. <https://doi.org/10.1145/322358.322363>
- Korth HK, Speegle G (1988) Formal model of correctness without serializability. ACM SIGMOD Rec 17: 379–386
- Lamport L (1976) Towards a theory of correctness for multi-user data base system. Tech. Rep., TRCA-7610-0712, Massachusetts Computer Associates
- Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. <https://doi.org/10.1145/359545.359563>
- Li C, Porto D, Clement A, Gehrke J, Preguiça NM, Rodrigues R (2012) Making geo-replicated systems fast as possible, consistent when necessary. In: OSDI, vol 12, pp 265–278
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don’t settle for eventual: scalable causal consistency for wide-area storage with cops. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP’11. ACM, New York, pp 401–416. <https://doi.org/10.1145/2043556.2043593>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Proceedings of the 10th USENIX conference on networked systems design and implementation, NSDI’13. USENIX Association, Berkeley, pp 313–328. <http://dl.acm.org/citation.cfm?id=2482626.2482657>
- Lynch NA (1983) Multilevel atomicity: a new correctness criterion for database concurrency control. ACM Trans Database Syst (TODS) 8(4):484–502
- Nawab F, Arora V, Agrawal D, El Abbadi A (2015) Charrots: a scalable shared log for data management in multi-datacenter cloud environments. In: Proceedings of the 18th international conference on extending database technology, EDBT 2015, Brussels, pp 13–24. <https://doi.org/10.5441/002/edbt.2015.03>
- Preguiça N, Marques JM, Shapiro M, Letia M (2009) A commutative replicated data type for cooperative editing. In: 29th IEEE international conference on distributed computing systems, ICDCS’09. IEEE, pp 395–403
- Roy S, Kot L, Bender G, Ding B, Hojjat H, Koch C, Foster N, Gehrke J (2015) The homeostasis protocol: avoiding transaction coordination through program analysis. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 1311–1326
- Satyaranayanan OT, Agrawal D (1993) Efficient execution of read-only transactions in replicated multiversion databases. IEEE Trans Knowl Data Eng 5(5):859–871. <https://doi.org/10.1109/69.243514>
- Schlageter G (1981) Optimistic methods for concurrency control in distributed database systems. In: Proceedings of the seventh international conference on very large data bases, vol 7, VLDB Endowment, VLDB’81, pp 125–130. <http://dl.acm.org/citation.cfm?id=1286831.1286844>

- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) Convergent and commutative replicated data types. *Bull Eur Assoc Theor Comput Sci* (104):67–88
- Shasha D, Llirbat F, Simon E, Valduriez P (1995) Transaction chopping: algorithms and performance studies. *ACM Trans Database Syst (TODS)* 20(3):325–363
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 385–400. <https://doi.org/10.1145/2043556.2043592>
- Weikum G (1985) A theoretical foundation of multi-level concurrency control. In: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on principles of database systems. ACM, pp 31–43
- Zhang Y, Power R, Zhou S, Sovran Y, Aguilera MK, Li J (2013) Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP'13. ACM, New York, pp 276–291. <https://doi.org/10.1145/2517349.2522729>

Co-resident Attack in Cloud Computing: An Overview

Sampa Sahoo, Sambit Kumar Mishra,
Bibhudatta Sahoo, and Ashok Kumar Turuk
National Institute of Technology Rourkela,
Rourkela, India

Introduction

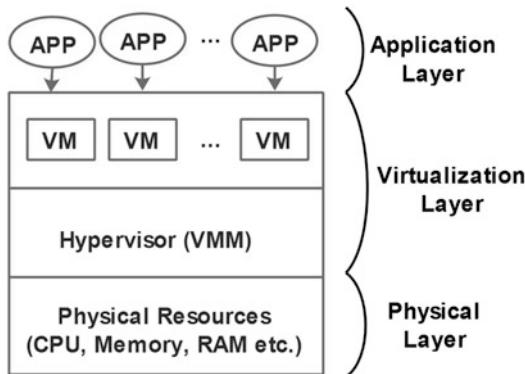
Cloud computing is a novel computing model that leverages on-demand provisioning of computational and storage resources. It uses pay-as-you-go economic model, which helps to reduce CAPEX and OPEX. Cloud computing essence and scope can be best described by the “5-3-2 principle,” where 5 stands for its essential characteristics (on-demand service, ubiquitous network access, resource pooling, rapid elasticity, and measured service) (Puthal et al. 2015). Three stands for service delivery models (SaaS, PaaS, and IaaS). Two points to basic deployment models (private and public) of the cloud (Sahoo et al. 2015).

Figure 1 shows the layered architecture of cloud. Various layers of a cloud are as follows:

application, virtualization, and physical. Application layer consists of user applications. The virtualization layer’s main components are a hypervisor or Virtual Machine Manager (VMM) and VM. A hypervisor acts as an administrator for the virtualized data center that permits the user to configure and manage cloud resources to create and deploy virtual machines and services in a cloud. A VM is an emulation of a physical machine. The virtualization technology allows creating several VMs on a single physical machine (Mishra et al. 2018a). The physical layer is a container of various physical resources like memory, RAM, etc. (Sahoo et al. 2016).

The rise of multimedia data and ever-growing requests for multimedia applications has tendered multimedia as the biggest contributor of big data. Applications on entertainment demand low security as compared to personalized videos like business meetings, telemedicine, etc. A secured video transmission ensures authorized entrance and prevents information leakage by any unapproved eavesdroppers. Various cryptographic algorithms (AES, DES, etc.) can be used to guarantee security to the multimedia application. Few examples of big data are credit card transactions, Walmart customer transactions, and data generated by Facebook users. A security breach in big data will cause severe legal consequences and reputational damage than at present. The use of big data in business also gives rise to privacy issues (Inukollu et al. 2014). Big data applications demand scalable and cost-efficient technology for huge data storage, computation, and communication. One of the solutions to address these issues is the use of cloud computing which is both scalable and cost-efficient. Since cloud computing is an umbrella term used by many technologies like networks, databases, virtualization, operating system, resource scheduling, transmission management, load balancing, etc., security issues related to these systems are also applicable to cloud computing.

Cloud computing facilitates business models and allows renting of IT resources on a utility like a basis through the Internet. Security is one of the most challenging issues to deal with in cloud computing due to numerous forms of



Co-resident Attack in Cloud Computing: An Overview, Fig. 1 Layered architecture of cloud

attacks on the application side as well as the hardware components. Various reasons for difficulties correlated with cloud computing are the loss of control, lack of trust, and multi-tenancy (Bhargava 2010). A cloud user outsourced its content to a remote server, losing direct control over the data, which becomes one of the reasons for cloud security issues. An attacker can target the communication link between cloud provider and clients. Multi-tenants of cloud use a shared pool of resources having conflicting goals. This gives opportunities to an attacker to steal useful information from the legitimate user on the same physical machine.

The virtualization method allows sharing of computing resources among many tenants, which may be business partners, suppliers, competitors, or attackers (Mishra et al. 2018b). Even though there is substantial logical isolation between VMs, shared hardware creates vulnerabilities to side channel attacks, i.e., data leakage through implicit channels (Zhang et al. 2011). The following sections discuss security issues in the cloud, co-resident attack, and its defense methods, followed by the conclusion.

Security Issues in Cloud

Security is a dominant barrier to the development and widespread use of cloud computing. Big data computing in cloud environment fur-

ther intensified security and privacy issues. The cheaper compute environment, networked application environment, and shared cloud system environment generate security, compression, encryption, and access control challenges, which must be addressed in a systematic way (Cloud-Security-Alliance 2012). Cloud provider security implementation mainly focused on the assumption that outsiders of the cloud are evil, whereas insiders are good. An outside attacker performs the following actions to get unauthorized access: listen to network traffic, insert malicious traffic, and launch DoS. An inside attacker can log on to user communication, peek into VMs or make copies of VMs, and monitor application pattern for unapproved access. The deficit security features in local devices also provide a way for malicious services on the cloud, as an attacker can use these devices to attack local networks.

Various terms representing the security or privacy attributes are confidentiality, integrity, availability, accountability, and privacy preservability (Xiao and Xiao 2013). Confidentiality ensures that user's data and computation are kept confidential from both the cloud provider and other users. Primary threats to confidentiality are co-resident VM attack and malicious system admin. Cloud integrity implies that data are stored honestly on data storage, computation is performed without any distortion, and integrity is affected by data loss or manipulation and dishonest computation. Availability ensures that cloud services are available most of the time and meet the SLA. It is influenced by flooding attack (denial of service (DoS)) and fraudulent resource consumption (FRC) attack. Accountability implies capability of identifying a party responsible for the specific event. Several threats to cloud accountability are SLA violation, inaccurate billing, etc. Since cloud user's data and business logic reside among distributed cloud servers, there are potential risks that confidential and personal information will be disclosed to public or business competitors. Privacy preservation is one of the critical attributes of privacy, which prevents disclosure of private data. Confidentiality is indispensable, whereas integrity guarantees

data, or computation is corruption-free, which somehow preserves privacy.

The following challenges are faced by cloud providers to build secure and trustworthy cloud system.

- *Outsourcing*: A cloud user outsourced its content to the cloud provider's data center and lost the direct physical control over the data. The computing and data storage are done at the cloud provider's end, causing the main reason for cloud insecurity. Outsourcing may also incur privacy violations.
- *Multi-tenancy*: Since multiple customers share the cloud resources, it gives rise to co-residency issues. Cross-VM or co-resident VM attack exploits the multi-tenancy nature.
- *Massive data and intense computation*: Traditional security methods may not suffice for massive data storage and computation.

Various Attacks on VM

VMs are the principal computing unit in the cloud computing environment that facilitate efficient utilization of hardware resources and reduce maintenance overhead of computing resources. VMs provide the additional layer of hardware abstraction and isolation but require quality methods to prevent attackers to access information from other VMs and the host. Vulnerabilities in VMs pose an immediate threat to the privacy and security of the cloud services. The shared and distributed nature of cloud resources makes it difficult to develop a security standard that ensures data security and privacy. Vulnerabilities in the cloud are the security loopholes, which allow unauthorized access to the network and infrastructure resources. Vulnerabilities in virtualization or hypervisor will enable an attacker to perform a cross-VM side-channel attack, and DoS attack provides access to legitimate user's VM. Solution directives for VM attacks include alert message for any breach of VMs isolation, integrity and security assurance of VM images, vulnerability-free virtual machine manager (VMM) or hypervisor, etc. (Modi et al.

2013). Different classes of attacks (Hyde 2009) that affect VMs are:

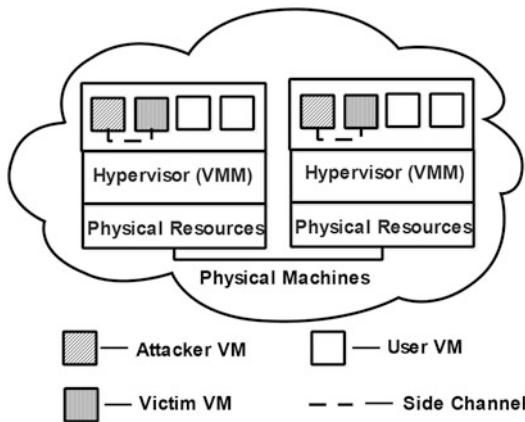
- *Co-resident attack*: Attacker adversary VM to communicate with other colocated VMs on the same host, thus violating isolation feature of VMs.
- *Attack on hypervisor*: The attack on a hypervisor allows the attacker to access VM, host operating system, and hardware.
- *DoS attack*: DoS attacks are endeavors by an unapproved user to degrade or deny resources to a genuine user. DoS attack consumes resources from all VMs on the host.

C

Co-resident Attack

Colocated VMs are logically separated from each other. Users are migrating to the cloud that was exposed to an additional threat caused by neighbors due to the sharing of resources. This exposure raises questions about neighbors' trustworthiness and integrity as a malicious user can build side channel to circumvent logical isolation and extract sensitive information from co-resident VMs (Han et al. 2017). Like traditional web services, VMs are exposed to various security threats, and co-resident attack is one of them. Other names of co-resident attack are co-residence, co-residency, and colocation attack. Figure 2 shows co-resident VM attack in cloud. It shows that attacker VM and target VM are placed on the same physical machine, where attacker extracts target VM information through the side channel.

In co-resident attack, malicious users extract information from colocated virtual machines on the same physical machine by building side channels. The various side-channel attack includes cache attack, timing attack, power monitoring attack, software-initiated fault attack, etc. In cache attack, attackers take action based on cache access of the victim in a virtualized environment like a cloud, whereas timing attack is based on computation time of various activities (e.g., the comparison between attacker's password with



Co-resident Attack in Cloud Computing: An Overview, Fig. 2 Co-resident attack

victim's unknown one). The power monitoring attack makes use of power consumption variation by the hardware during computation. Software-initiated fault attack is a rare one, where off-limits memory is changed by accessing adjacent memory too often.

The co-resident attack is a two-step process. First, the attacker sets target VMs and collocates their VMs with these VMs on the corresponding physical servers. The second process builds the side channels to extract valuable information from the victim. The following steps are performed by an attacker for co-resident attack (Han et al. 2016):

1. An attacker first targets some VMs.
2. Attacker starts some VMs either from a single account or multiple accounts.
3. Check whether new VM is colocated with target VM on the same physical machine or not.
4. Repeat steps 2–3 until co-residency is achieved.
5. If co-residency is achieved, non-colocated VMs are turned off. Now, the attacker gains access and steals information from the target VM.

Defense Methods

Cloud security providers mainly aim to prevent the attacker from achieving co-residence. Co-

residence may lead to cross-VM information leakage. The attacker analyzes the cloud infrastructure and identifies the residency of target VM. After several attempts, an attacker may succeed to instantiate new VMs, co-resident with target VM. Now, the attacker can leak all kinds of information which are known as the side-channel attack. This is because both target and attacker VM share the same resource. Various methods used to defend co-resident attack in cloud are listed below:

- *Elimination of side channels:* One of the solutions to handle co-resident attack is the elimination of side channels. But, this method needs modifications to the current cloud platform.
- *VM-allocation policy:* Another answer is VM allocation policy, which can influence the probability of colocation. The bigger the probability of non-co-residency, the better the policy is.
- *Increasing difficulty level of co-residence verification:* Usually the IP address of a VM Dom0 is a privileged VM responsible for management of all VMs on a host. For co-resident VMs, Dom0 IP addresses must be same. So, the disclosure of Dom0 IP address must be restricted.
- *Detection of co-resident attacks trademark:* Any attack can be identified by unusual behavior of a cloud system. Various indicators of co-resident attack include abnormalities in the CPU and RAM behavior, cache miss, system calls, etc.
- *Periodic VM migration:* Periodic migration of VMs helps to counter co-resident attack. Like every coin has two sides, this method comes with additional drawbacks like extra power consumption, performance degradation, SLA violation, etc.
- *Encrypted VM images:* Encrypted VM images can guarantee the security of a VM, which can be accessible by only an authorized user who knows the security keys of permitted access.

Case Study

Multiple VMs of various organizations run on Amazon's EC2 infrastructures with virtual boundaries, and each VM can operate within a single physical server. VMs located on the same physical server have IP addresses approaching each other. An attacker can set up lots of his own virtual machines to get a glance at their IP addresses and figure out VMs that share the same resources as a meditated target. Once the adversary VM is located along with target VM on the same server, an attacker can deduce sensitive information about the victim, and this action can cause resource usage fluctuations.

Conclusion

This chapter manifests various security and privacy issues of cloud running big data applications. Virtualization technology and VM are the two most essential components of a cloud. Our aim in this paper is to give an overview of various security issues in the cloud (specific to VM) and its solutions. Here, we present details about co-resident VM attack that includes steps performed by an attacker, various defense methods, and the explanation with an example.

References

- Bhargava B, Cho Y, Kim A (2010) Research in cloud security and privacy. <https://www.cs.purdue.edu/homes/bb/cloud/Security-Privacy.ppt>
- Cloud-Security-Alliance (2012) Top ten big data security and privacy challenges. https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Top_Ten_v1.pdf
- Han Y, Alpcan T, Chan J, Leckie C, Rubinstein BIP (2016) A game theoretical approach to defend against co-resident attacks in cloud computing: preventing co-residence using semi-supervised learning. *IEEE Trans Inf Forensics Secur* 11(3):556–570
- Han Y, Chan J, Alpcan T, Leckie C (2017) Using virtual machine allocation policies to defend against co-resident attacks in cloud computing. *IEEE Trans Dependable Secure Comput* 14(1):95–108
- Hyde D (2009) A survey on the security of virtual machines. www.cse.wustl.edu/jain/cse571-09/ftp/vmsec/index.html

Inukollu VN, Arsi S, Ravuri SR (2014) Security issues associated with big data in cloud computing. *Int J Netw Secur Appl* 6(3):45

Mishra SK, Putha D, Rodrigues JJ, Sahoo B, Dutkiewicz E (2018a, in press) Sustainable service allocation using metaheuristic technique in fog server for industrial applications. *IEEE Trans Ind Inf*

Mishra SK, Puthal D, Sahoo B, Jena SK, Obaidat MS (2018b) An adaptive task allocation technique for green cloud computing. *J Supercomput* 74(1):370–385

Modi C, Patel D, Borisaniya B, Patel A, Rajarajan M (2013) A survey on security issues and solutions at different layers of cloud computing. *J Supercomput* 63(2):561–592

Puthal D, Sahoo B, Mishra S, Swain S (2015) Cloud computing features, issues, and challenges: a big picture. In: Computational Intelligence and Networks (CINE), 2015 International Conference on, IEEE, pp 116–123

Sahoo S, Nawaz S, Mishra SK, Sahoo B (2015) Execution of real time task on cloud environment. In: India Conference (INDICON), 2015 Annual IEEE, IEEE, pp 1–5

Sahoo S, Sahoo B, Turuk AK, Mishra SK (2016) Real time task execution in cloud using mapreduce framework. In: Resource management and efficiency in cloud computing environments, IGI Global, p 190

Xiao Z, Xiao Y (2013) Security and privacy in cloud computing. *IEEE Commun Surv Tutorials* 15(2):843–859

Zhang Y, Juels A, Oprea A, Reiter MK (2011) Home-alone: co-residency detection in the cloud via side-channel analysis. In: 2011 IEEE Symposium on Security and Privacy, pp 313–328

CRUD Benchmarks

Steffen Friedrich and Norbert Ritter
Department of Informatics, University of Hamburg, Hamburg, Germany

Synonyms

[NoSQL benchmarks](#)

Definitions

A CRUD benchmark is a generic experimental framework for characterizing and comparing the performance of database systems by executing

workloads using only a simple CRUD (create, read, update, and delete) interface.

Since each database system provides at least this minimal subset of operations, CRUD benchmarks allow a general performance comparison of a variety of non-relational *NoSQL database systems*.

Overview

For many decades, relational database management systems (RDBMSs) have been the first choice for business applications. A high degree of standardization, especially the standard query language SQL, made it possible to develop domain-specific benchmarks for RDBMSs.

The DebitCredit benchmark by Jim Gray laid down the foundation for standardized database benchmarks (Anon et al. 1985). It introduced most of the key ideas that were later taken up by the Transaction Processing Performance Council (TPC) for the specification of the TPC-A benchmark and its famous successor, TPC-C (2010). The TPC was founded in 1988 as an industry-consensus body to fulfill the need for benchmark certification. Previously, many RDBMS vendors had conducted benchmarks based on their own interpretation of DebitCredit. Since its foundation, the TPC has developed numerous benchmarks for various kinds of online transactional processing (OLTP) and also online analytical processing (OLAP) workloads.

However, the amount of useful data in some applications has become so vast that it cannot be stored or processed by traditional relational database solutions (Hu et al. 2014). At the beginning of the twenty-first century, this led to the development of non-relational database systems able to cope with such Big Data. These systems are subsumed under the term *NoSQL database systems*. They offer horizontal scalability and high availability by sacrificing querying capabilities, the strict ACID guarantees, and the transactional support as known from RDBMSs.

Since most TPC benchmark specifications assume ACID compliant transactions, they cannot be applied to mostly non-transactional NoSQL

databases. Defining similar domain-specific standards for those systems is even more complicated due to the sheer heterogeneity in their data models and APIs.

Therefore, many NoSQL vendors and open source communities implemented their own database-specific performance measurement tools, just as the RDBMS vendors did before TPC. These include, for example, the Cassandra-stress tool, Mongoperf for MongoDB, the Redis-benchmark utility, or Basho_bench for Riak. In the end, the results of these tools are often difficult to understand, cannot be extended to other scenarios, and do not allow comparisons between different databases.

Eventually, the *Yahoo! Cloud Serving Benchmark* (YCSB) was proposed (Cooper et al. 2010). YCSB is limited to a simple CRUD interface and has become the de facto standard for NoSQL database performance comparison, which is why the term *NoSQL Benchmark* is often used interchangeably for CRUD benchmark. Due to the fact that YCSB is open source and extensible, the developer community has added support for almost every NoSQL database system since the release in 2010.

Scientific Fundamentals

The Benchmark Handbook for Database and Transaction Systems edited by Jim Gray (1993) can be regarded as the first reference work of database benchmarking. It identifies four requirements that a domain-specific benchmark must meet:

1. **Relevance** means that the benchmarks should be based on a realistic workload such that the results reflect something important to their readers.
2. **Simplicity** describes that the workload should be understandable.
3. **Portability** means that it should be easy to run the benchmark against a large number of systems.

4. **Scalability** says that it should be possible to scale the benchmark up to larger systems as performance and architectures evolve.

These requirements have been refined over time to reflect the current developments of database systems (Huppler 2009; Folkerts et al. 2013). The book *Cloud Service Benchmarking* by Bermbach et al. (2017) is an excellent modern reference work in which these design objectives are also discussed in detail with regard to cloud computing and Big Data requirements. Regarding the mentioned four requirements, CRUD benchmarks significantly weaken the relevance aspect in order to achieve the greatest possible portability, simplicity, and scalability. They do not model a real domain-specific application workload like the TPC standard specifications but a simpler one in which the choice of CRUD operations is selected according to a given probability distribution of real web application workloads (*workload generation*). The typical *metrics* measured during the benchmark run are throughput (operations per second) and latency (response time in milliseconds). CRUD benchmarks also define hardly any restrictions on the *system under test* in favor of portability. The actual data values consist of simple synthetically generated random character strings (*data generation*). Overall, this makes them less relevant for concrete applications but allows to measure the performance of highly scalable NoSQL systems in the first place.

Hence, CRUD benchmarks are also often used and extended in research to measure additional performance aspects of distributed NoSQL database systems. An overview of these research efforts can be found in Friedrich et al. (2014), and the book by Bermbach et al. (2017) can also be recommended as a reference for the current state of measurement methods for nonfunctional properties such as scalability, elasticity, and consistency of NoSQL data management solutions.

It should be noted that the basics of scalability and elasticity benchmarking were already determined by the Wisconsin benchmark, an early performance evaluation framework for RDBMS (DeWitt 1993). It was by no means as popular as

DebitCredit and the TPC benchmarks for comparative studies by RDBMS vendors, but it was widely used in research for measuring the performance of parallel database system prototypes designed for shared nothing hardware architectures. As these database prototypes should already meet horizontal scalability and elasticity requirements, the benchmark defined the key metrics scale-up and speedup to measure them.

Current research work is further developing domain-specific benchmarks for certain classes of NoSQL database systems such as document stores that go beyond pure CRUD benchmarks. This allows to compare the performance of more sophisticated query capabilities in the context of a particular application. Reniers et al. (2017) give an up-to-date overview of these domain-specific NoSQL benchmarks and CRUD benchmarks.

Key Applications

In addition to the four requirements that a benchmark must meet, Gray (1993) also describes four motivations for database benchmarking which can be extended to NoSQL database performance evaluation:

1. **Different proprietary software and hardware systems:** The classical competitive situation between two relational database vendors running their solution on their own dedicated specialized hardware can, nowadays, for example, be transferred to proprietary NoSQL database as a service solutions (DBaaS) as provided by cloud service providers.
2. **Different software products on the same hardware:** This is the most common form in which NoSQL database system vendors compare their systems with others. The different databases are set up in a cluster of the same hardware or cloud infrastructure, and their performance is usually measured with a selection of different workloads.
3. **The same software product on different hardware systems:** The database workload serves as a reference application for comparing different underlying hardware platforms.

Nowadays, this can mean that users benchmark their NoSQL database system on different infrastructure as a service (IaaS) solutions to select the most performant or cost-effective cloud provider.

4. **Different releases of a software product on the same hardware:** The benchmarking of newer versions of a database system is an integral part of the modern agile software development process. This is one of the major reasons why NoSQL developers often implement database-specific performance measurement tools.

Comparative studies are primarily promoted by database vendors themselves for marketing and advertising purposes. Since there is no instance like TPC for the verification of benchmark results in the NoSQL area, these results should always be looked at critically, because the workload parameters, the hardware, and the presentation of the results are selected in such a way that the own product performs particularly well. This kind of *benchmarketing* already existed for RDBMSs in the early 1980s.

In addition to comparative studies, one of the main motivations for benchmarking is systems research, i.e., understanding the performance behavior of certain system designs or understanding quality trade-offs like the consistency/latency trade-off of different data replication techniques.

Future Directions

The research in the field of benchmarking nonfunctional properties of distributed NoSQL database systems is still in an early stage. For example, a number of methods for measuring the consistency of replicated databases have already been proposed. Apart from further studies on scalability and elasticity, there is only little work on the topic of availability benchmarking of NoSQL database systems, especially not under different failure conditions. Therefore, further research in this area is still needed to develop

standard methodologies for measuring these requirements.

Furthermore, there is a variety of possible benchmark designs between simple CRUD benchmarks and application-oriented benchmarks for one specific NoSQL data model that still need to be explored. One can imagine domain-specific workloads that are still general enough to allow specific implementations for all the different data models.

Cross-References

- [NOSQL Database Systems](#)
- [System Under Test](#)
- [YCSB](#)

References

- Anon, Bitton D, Brown M, Catell R, Ceri S, Chou T, DeWitt D, Gawlick D, Garcia-Molina H, Good B, Gray J, Homan P, Jolls B, Lukes T, Lazowska E, Nauman J, Pong M, Spector A, Trieber K, Sammer H, Serlin O, Stonebraker M, Reuter A, Weinberger P (1985) A measure of transaction processing power. Datamation 31(7):112–118
- Bermbach D, Wittern E, Tai S (2017) Cloud service benchmarking: measuring quality of cloud services from a client perspective. Springer, Cham
- Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, SoCC’10. ACM, New York, pp 143–154
- DeWitt DJ (1993) The wisconsin benchmark: past, present, and future. In: Gray J (ed) The benchmark handbook. Morgan Kaufmann, San Mateo
- Folkerts E, Alexandrov A, Sachs K, Iosup A, Markl V, Tosun C (2013) Benchmarking in the cloud: what it should, can, and cannot be. Springer, Berlin/Heidelberg, pp 173–188
- Friedrich S, Wingerath W, Gessert F, Ritter N (2014) NoSQL OLTP benchmarking: a survey. In: 44. Jahrestagung der Gesellschaft für Informatik, Informatik 2014, Big Data – Komplexität meistern, 22–26 Sept 2014 in Stuttgart, pp 693–704
- Gray J (ed) (1993) The benchmark handbook for database and transaction systems, 2nd edn. Morgan Kaufmann, San Mateo

- Hu H, Wen Y, Chua TS, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. *IEEE Access* 2:652–687
- Huppler K (2009) The art of building a good benchmark. Springer, Berlin/Heidelberg, pp 18–30
- Reniers V, Van Landuyt D, Rafique A, Joosen W (2017) On the state of NoSQL benchmarks. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion, ICPE'17 companion. ACM, New York, pp 107–112
- TPC-C (2010) Benchmark specification. <http://www.tpc.org/tpcc>

Cryptocurrency

► [Blockchain Transaction Processing](#)

D

Data Archives

- ▶ [Data Longevity and Compatibility](#)

Data Archiving

- ▶ [Computing the Cost of Compressed Data](#)

Data Cleaning

Xu Chu

School of Computer Science, Georgia Institute
of Technology, Atlanta, GA, USA

Synonyms

[Data cleansing](#)

Definitions

Data cleaning is used to refer to all kinds of tasks and activities to detect and repair errors in the data.

Overview

Enterprises have been acquiring large amounts of data from a variety of sources to build their own “data lakes,” with the goal of enriching their data asset and enabling richer and more informed analytics. Data collection and acquisition often introduce errors in data, for example, missing values, typos, mixed formats, replicated entries for the same real-world entity, outliers, and violations of business rules.

A Kaggle’s 2017 survey about the state of data science and machine learning reveals that dirty data is the most common barrier faced by workers dealing with data (Kaggle 2017). Not surprisingly, developing effective and efficient data cleaning solutions is a challenging venue and is rich with deep theoretical and engineering problems.

There are various surveys and books on different aspects of data quality and data cleaning. Rahm and Do (2000) give a classification of different types of errors that can happen in an extract-transform-load (ETL) process and survey the tools available for cleaning data in an ETL process; Bertossi (2011) provides complexity results for repairing inconsistent data and performing consistent query answering on inconsistent data; Hellerstein (2008) focuses on cleaning quantitative data, such as integers and floating points, using mainly statistical outlier detection techniques; Fan and Geerts (2012) discuss the use of data quality rules in data consistency, data currency, and data completeness and how different

aspects of data quality issues might interact; Dasu and Johnson (2003) summarize how techniques in exploratory data mining can be integrated with data quality management; Ilyas and Chu (2015) provide taxonomies and example algorithms of qualitative error detection and repairing techniques; and Ganti and Das Sarma (2013) focus on an operator-centric approach for developing a data cleaning solution, which involves the development of customizable operators that could be used as building blocks for developing common solutions.

This chapter covers four commonly encountered data cleaning tasks, namely, outlier detection, rule-based data cleaning, data transformation, and data deduplication. Since there is a very large body of work on these tasks, this chapter only intends to provide an introduction to each data cleaning task and categorize various techniques proposed in the literature to tackle each task. There are also many complementary research efforts to data cleaning that are not covered in this chapter, such as data profiling, provenance, and metadata management, data discovery, consistent query answering, metadata management, schema mapping, and data exchange.

Key Research Findings

Outlier Detection

Outlier detection refers to the activity of detecting outlying values and is a quantitative error detection task. While an exact definition of an outlier depends on the application, there are some commonly used definitions, such as “an outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism” (Hawkins 1980). For example, for a company whose employees’ salaries are mostly around 100 K, an employee with a salary of 10 K can be considered to be an outlying record.

Multiple surveys and tutorials have been published to summarize different definitions of outliers and algorithms for detecting them (Hodge and Austin 2004; Chawla and Sun 2006; Aggarwal 2013). In general, outlier

detection techniques fall into three categories: statistics-based, distance-based, and model-based. Statistics-based outlier detection techniques assume that the normal data points would appear in high-probability regions of a stochastic model, while outliers would occur in the low-probability regions of a stochastic model (Grubbs 1969). They can often provide a statistical interpretation for discovered outliers, or a score/confidence interval for a data point being an outlier, rather than making a binary decision. Distance-based outlier detection techniques often define a distance between data points, which is used for defining a normal behavior, for example, normal data point should be close to a lot of other data points, and data points that deviate from such normal behavior are declared outliers (Knorr and Ng 1998). A major advantage of distance-based techniques is that they are unsupervised in nature and do not make any assumptions regarding the generative distribution for the data. Instead, they are purely data driven. Model-based outlier detection techniques first learn a classifier model from a set of labeled data points and then apply the trained classifier to a test data point to determine whether it is an outlier (De Stefano et al. 2000). Model-based approaches assume that a classifier can be trained to distinguish between the normal data points and the anomalous data points using the given feature space. They label data points as outliers if none of the learned models classify them as normal points.

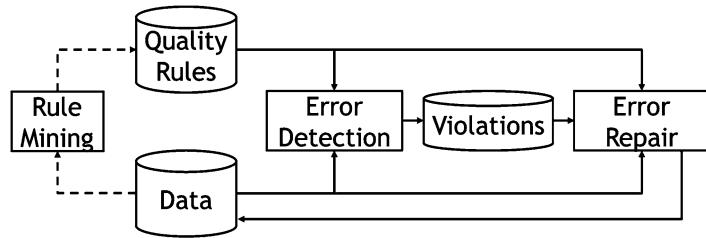
Enforcing Data Quality Rules

One common way to detect and rectify errors in databases is through the enforcement of data quality rules, often expressed as integrity constraints (ICs) (Chu et al. 2013; Fan and Geerts 2012). In this context, (qualitative) error detection is the process of identifying *violations of ICs*, namely, subsets of records that cannot co-exist together; and error repair is the exercise of modifying the database, such that the violations are resolved and the new data conforms to the given ICs.

Figure 1 shows a typical workflow of qualitative data cleaning. In order to clean a dirty dataset using qualitative data cleaning techniques, there

Data Cleaning, Fig. 1

Rule-based data cleaning workflow with an optional rule mining step, the error detection step, and the error repair step



D

need to be data quality rules that reflect the semantics of the data. One way to obtain data quality rules is by consulting domain experts, which requires a lot of domain expertise and is usually a time-consuming processing. An alternative approach is to design an algorithm to automatically discover data quality rules. Given a dirty dataset and the associated data quality rules, the error detection step detects violations of the specified rules in the data. A violation is *minimal set of cells in a dataset that cannot coexist together*. Finally, given the violations and the rules that generate those violations, the error repair step produces data updates, which are applied to the dirty dataset. The error detection and the error repair loop go on until the data conforms to the data quality rules, namely, there is no violations in the data.

Example 1 Consider Table 1 that contains employee records in a company. Every tuple specifies a person in a company with her id (GID), name (FN, LN), level (LVL), zip code (ZIP), state (ST), and salary (SAL). Suppose two data quality rules hold for this table. The first rule states that, if two employees have the same zip code, they must have the same state. The second rule states that among employees working in the same state, a higher-level employee cannot earn less salary than a lower-level employee.

Given these two data quality rules, the error detection step detects two violations. The first violation consists of four cells $\{t_1[\text{ZIP}], t_1[\text{ST}], t_3[\text{ZIP}], t_3[\text{ST}]\}$, which together violate the first data quality rules. The second violation consists of six cells $\{t_1[\text{ROLE}], t_1[\text{ST}], t_1[\text{SAL}], t_2[\text{ROLE}], t_2[\text{ST}], t_2[\text{SAL}]\}$, which together violate the second data quality rule. The data repair step takes the violations and produces an

Data Cleaning, Table 1 An example employee table

TID	FN	LN	LVL	ZIP	ST	SAL
t_1	Anne	Nash	5	10001	NM	110,000
t_2	Mark	White	6	87101	NM	80,000
t_3	Jane	Lee	4	10001	NY	75,000

update that changes $t_1[\text{ST}]$ from “NM” to “NY,” and the new data now has no violation with respect to the two rules.

Data Transformation

Data transformation refers to the task of transforming data from one format to another. Data transformation tasks can be classified into two types in general: *syntactic data transformations* and *semantic transformations*. Syntactic transformations aim at transforming a table from one syntactic format to another, often without requiring external knowledge or reference data. On the other hand, semantic transformations usually involve understanding the meaning/semantics, instead of simply as a sequence of characters; hence, they usually require referencing external data sources.

Example 2 Figure 2a shows a syntactic transformation, where a tabular data containing person names, telephone numbers, and fax numbers, originally arranged in a tabular format with phone and fax numbers stored in the same column as a complex data type, are flattened into a normal relational table representation to enable efficient querying of this data. In addition, the telephone numbers and the fax numbers are also transformed by adding two dashes between the nine digits. Both cases do not require external knowledge to perform these transformations.



Data Cleaning, Fig. 2 Example transformations. (a) Syntactic transformation. (b) Semantic transformation

Figure 2b shows a semantic transformation, where country codes are transformed into country names; obviously, it requires external knowledge, such as an external knowledge base that contains both full and abbreviated country names.

There are generally three major components/dimensions of a syntactic data transformation system: *language*, *authoring*, and *execution*. Since there are many ways to transform a dataset, syntactic data transformation solutions usually adopt a *transformation language* that limits the space of possible transformations. The language could consist of a finite set of operations allowed on a table (Raman and Hellerstein 2001; Kandel et al. 2011), such as splitting a column and merging two columns, or it could consist of a set functions for string manipulations (Gulwani 2011), such as extracting substring and concatenating two substrings. There is a balance that needs to be achieved when choosing a particular language: it needs to be expressive enough so that it captures many real-world transformation tasks and, at the same time, restricted enough to allow for effective and easy authoring of the transformations. For a specific transformation task, syntactic transformation tools have different interaction models with the users to *author a transformation program* using the adopted language. The interaction models could be generally classified as declarative, example-driven, or proactive. In the declarative model,

users specify the transformations directly, usually through a visual interface. In the example-driven model, users are required to give a few input-output examples, based on which the tool automatically infers likely transformations that match the provided examples. In the proactive interaction model, the transformation tool provides hints or highlights for which data needs transformation and sometimes even suggests possible transformations so that users simply need to accept or reject the suggestions. Finally, *transformation execution* applies the specified transformations to the dataset. Since there might be multiple specified transformations from the previous step, transformation tools usually offer assistance to the users in selecting the desired transformation, for example, by displaying the effect of the specified transformation immediately on a sample data or by providing interpretable statements of the specified transformations. Different transformation tools offer different capabilities along these three dimensions.

Semantic transformations usually involve understanding the meaning/semantics, or the typical use of the data, instead of simply as a sequence of characters. In contrast to syntactic transformations, semantic transformations cannot be computed by solely looking at input values; rather, they usually require external data sources to look up the values that need to be transformed. Example-driven techniques have been mostly used for semantic transformations (Singh and Gulwani 2012; Abedjan et al. 2016). Users

are usually asked to provide a few input-output examples, and semantic transformation tools then will search the available external data sources to find a “query” that matches the examples. The query can then be used to transform new unseen data.

Data Deduplication

Data deduplication, also known as duplicate detection, record linkage, record matching, or entity resolution, refers to the process of identifying tuples in one or more relations that refer to the same real-world entity (Elmagarmid et al. 2007). A data deduplication process usually involves many steps and choices, including designing similarity metrics to evaluate the similarity for a pair of records, training classifiers to determine whether a pair of records are duplicates or not, clustering all records to obtain clusters of records that represent the same real-world entity, consolidating clusters of records to unique representations, designing blocking or distributed strategies to scale up the deduplication process, and involving humans to decide whether a record pair are duplicates when machines are uncertain.

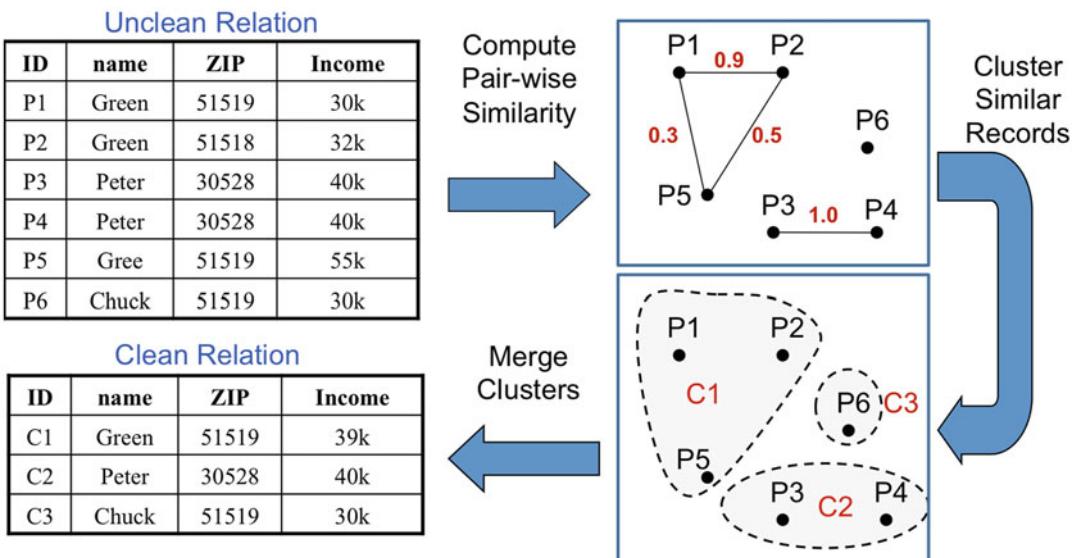
D

Example 3 Figure 3 illustrates a typical example of data deduplication. The similarities between pairs of records are computed and are shown in the similarity graph (upper right graph in Fig. 3). The missing edges between any two records indicate that they are non-duplicates. Records are then clustered together based on the similarity graph. Suppose the user sets the threshold to be 0.5, namely, any record pairs having similarity greater than 0.5 are considered duplicates. Although Records P_1 and P_5 have similarity less than 0.5, they are clustered together due to transitivity; that is, they are both considered duplicates to Record P_2 . All records in the same cluster are consolidated into one record in the final clean relation.

Applications of Data Cleaning

Data cleaning is a necessary step in many data-driven analytics. Different data cleaning tasks target different types of errors.

Applications of outlier detection include network intrusion detection, financial fraud detection, and abnormal medical condition



Data Cleaning, Fig. 3 A typical deduplication task

detection. For example, in the context of computer networks, different kinds of data, such as operating system calls and network traffic, are collected in large volumes. Outlier detection can help with detecting possible intrusions and malicious activities in the collected data.

Rule-based data cleaning can help clean any relational databases where data quality rules can be defined. The richer the semantics of the data is, the better rule-based data cleaning techniques are at detecting and repairing violations.

Data transformations are used in a variety of tasks and at different stages of the ETL life cycle. For example, before running a data integration project, transformations are often used to standardize data formats, to enforce standard patterns, or to trim long strings. Transformations are also used at the end of the ETL process, for example, to merge clusters of duplicate records, to find a unique representation for a cluster of records (aka golden record), or to prepare data to be consumed by analytics tools. Data transformations can also be seen as a tool for data repair in rule-based data cleaning, since it can be used to “transform” erroneous data.

Duplicate records can occur due to many reasons. For example, a customer might be recorded multiple times in a customer database if the customer used different names at the time of purchase; a single item might be represented multiple times in an online shopping website; and a record might appear multiple times after a data integration project because that record had different representations in original data sources. Data deduplication targets specifically duplicate records and resolves them.

Future Directions for Research

Data cleaning is a complicated process, and an end-to-end data cleaning solution usually involves many different cleaning sub-tasks. We have discussed techniques for tackling four common data cleaning tasks. There are still many challenges and opportunities in building practical data cleaning systems: (1) the scale of data renders many data cleaning techniques

insufficient. New cleaning solutions must adapt to growing datasets of the Big Data era, for example, by leveraging sampling techniques or distributed computation. (2) Although there are existing research about involving humans to perform data deduplication, for example, through active learning (Tejada et al. 2001; Sarawagi and Bhamidipaty 2002; Arasu et al. 2010), involving humans in other data cleaning tasks, such as repairing IC violations and taking user feedback in discovering of data quality rules, is yet to be explored; (3) a significant portion of data is residing in semi-structured formats (e.g., JSON) and unstructured formats (e.g., text documents). Data quality problems for semi-structured and unstructured data remain largely unexplored; and (4) there is significant concerns about data privacy as increasingly more individual data are collected by governments and enterprises. Data cleaning is by nature a task that requires examining and searching through raw data, which may be restricted in some domains including finance and medicine. How to perform most data cleaning tasks, while preserving data privacy, remains an open challenge.

Cross-References

- ▶ [Data Quality and Data Cleansing of Semantic Data](#)

References

- Abedjan Z, Morcos J, Ilyas IF, Ouzzani M, Papotti P, Stonebraker M (2016) Dataxformer: a robust transformation discovery system. In: Proceedings of 32nd international conference on data engineering, pp 1134–1145
- Aggarwal CC (2013) Outlier analysis. Springer, New York
- Arasu A, Götz M, Kaushik R (2010) On active learning of record matching packages. In: Proceedings of ACM SIGMOD international conference on management of data, pp 783–794
- Bertossi LE (2011) Database repairing and consistent query answering. Morgan & Claypool Publishers, San Rafael
- Chawla S, Sun P (2006) Outlier detection: principles, techniques and applications. In: Advances in knowledge discovery and data mining, 10th Pacific-Asia conference

- Chu X, Ilyas IF, Papotti P (2013) Holistic data cleaning: putting violations into context. In: Proceedings of 29th international conference on data engineering, pp 458–469
- Dasu T, Johnson T (2003) Exploratory data mining and data cleaning. Wiley, Hoboken
- De Stefano C, Sansone C, Vento M (2000) To reject or not to reject: that is the question—an answer in case of neural classifiers. *IEEE Trans Syst Man Cybern* 30(1):84–94
- Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: a survey. *IEEE Trans Knowl Data Eng* 19(1):1–16
- Fan W, Geerts F (2012) Foundations of data quality management. Synthesis lectures on data management. Morgan & Claypool Publishers, San Rafael
- Ganti V, Sarma AD (2013) Data cleaning: a practical perspective. *Synth Lect Data Manag* 5(3):1–85
- Grubbs FE (1969) Procedures for detecting outlying observations in samples. *Technometrics* 11(1):1–21
- Gulwani S (2011) Automating string processing in spreadsheets using input-output examples. In: Proceedings 38th ACM SIGACT-SIGPLAN symposium on principles of programming languages, pp 317–330
- Hawkins D (1980) Identification of outliers, vol 11. Chapman and Hall, London
- Hellerstein JM (2008) Quantitative data cleaning for large databases. United Nations Economic Commission for Europe (UNECE)
- Hodge VJ, Austin J (2004) A survey of outlier detection methodologies. *Artif Intell Rev* 22(2):85–126
- Ilyas IF, Chu X et al (2015) Trends in cleaning relational data: consistency and deduplication. *Found Trends® Databases* 5(4):281–393
- Kaggle (2017) <https://goo.gl/ZAZGsD>
- Kandel S, Paepcke A, Hellerstein J, Heer J (2011) Wrangler: interactive visual specification of data transformation scripts. In: Proceedings of SIGCHI conference on human factors in computing systems, pp 3363–3372
- Knorr EM, Ng RT (1998) Algorithms for mining distance-based outliers in large datasets. In: Proceedings of 24th international conference on very large data bases, pp 392–403
- Rahm E, Do HH (2000) Data cleaning: problems and current approaches. *IEEE Data Eng Bull* 23:2000
- Raman V, Hellerstein JM (2001) Potter's wheel: an interactive data cleaning system. In: Proceedings of 27th international conference on very large data bases, pp 381–390
- Sarawagi S, Bhamidipaty A (2002) Interactive deduplication using active learning. In: Proceedings of 8th ACM SIGKDD international conference on knowledge discovery and data mining, pp 269–278
- Singh R, Gulwani S (2012) Learning semantic string transformations from examples. *Proc VLDB Endow* 5(8):740–751
- Tejada S, Knoblock CA, Minton S (2001) Learning object identification rules for information integration. *Inf Syst* 26(8):607–633

Data Cleansing

► Data Cleaning

D

Data Compression

► Computing the Cost of Compressed Data

Data Consistency

► Database Consistency Models

Data Curation

► Data Quality and Data Cleansing of Semantic Data

Data Decay

► Data Longevity and Compatibility

Data Deduplication

► Large-Scale Entity Resolution

Data Differencing

► Delta Compression Techniques

Data Fusion

Jesús García, José Manuel Molina,
 Antonio Berlanga, and Miguel Angel Patricio
 Applied Artificial Intelligence Group,
 Universidad Carlos III de Madrid, Colmenarejo,
 Spain
 Computer Science and Engineering, Universidad
 Carlos III de Madrid, Colmenarejo, Spain

Introduction and Key Concepts of Information Fusion: Data, Models, and Context

Information fusion (IF) is a multi-domain-growing field aiming to provide data processes for situation understanding (Liggins et al. 2008). Globally, fusion systems aim to integrate sensor data and information/knowledge databases, contextual information, mission goals, etc., to describe dynamically changing situations. In a sense, the goal of information fusion is to obtain continuous refinements of estimates and assessments of a subset of the world based on partial observations and the evaluation of the need for additional sources or modification of the process itself, to achieve improved results.

The capability to fuse digital data and generate useful information is conditioned by the quality of inputs, whether device-derived or text-based. Data are generated in different formats, some of them unstructured and may be inaccurate, incomplete, ambiguous, or contradictory. The key aspect in modern DF applications (Gómez et al. 2011) is the appropriate integration of diverse types of information and knowledge (see Fig. 1): observational data, knowledge models (a priori or inductively learned), and contextual information. Each of these categories has a distinctive nature and potential support to the result of the fusion process.

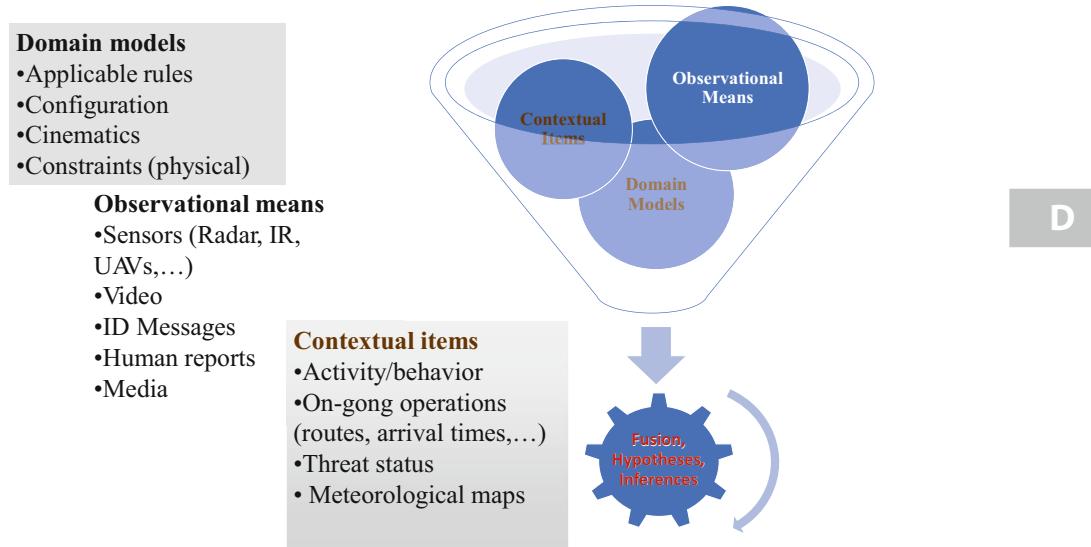
- **Observational data:** Observational data are the fundamental data about the dynamic scenario and entities of interest, as collected from the available observation means, including both physical devices (“hard” data)

and human observers (“soft” data) (Biermann et al. 2016).

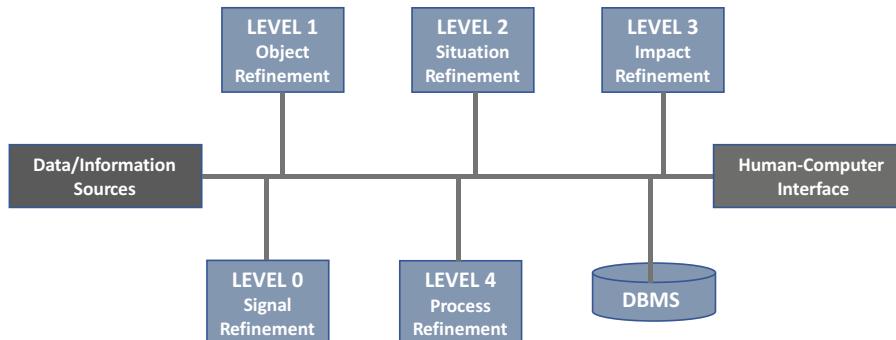
- **A priori and learned domain models:** Information fusion provides inferences about certain entities or hidden states (e.g., detection and tracking of certain objects, presence of a certain type of criminal activity, etc.). This involves combining data to update estimates, exploiting dynamic models of the expected behaviors of entities (physical models like kinematics or logical models with expected behaviors depending on context). Accurate models allow good predictions, being uncertainty modelling the key aspect to develop appropriate estimates, combining predictions and data observations. In those cases when a priori detailed models cannot be formed, one possibility is to try and excise the knowledge through online machine learning processes operating on the observational and other data. These are procedural and algorithmic methods for discovering relationships among and behaviors of entities of interest (Bishop 2004).
- **Context:** Contextual information has become fundamental to develop models in complex scenarios. Among other possible alternatives, context can be defined as “the set of circumstances surrounding a task that are potentially of relevance to its completion” (Bettini et al. 2010). Because of its task relevance, fusion or estimating/inferring task implies the development of a best possible estimate, considering this lateral knowledge (Snidaro et al. 2014, 2016). We can see the context as background, i.e., not the specific entity, event, or behavior of prime interest but that information which is influential to the formation of a best estimate of these items.

Levels of Abstraction: The JDL Model

Among different alternatives to classify IF functions and problems, the model developed by Joint Directors of Laboratories Data Fusion Subpanel has gained the greatest recognition. The “JDL model” differentiates functions into fusion “levels” that relate to the refinement of estimates



Data Fusion, Fig. 1 Information fusion. Observations, models, and context



Data Fusion, Fig. 2 JDL model

for parameters of interest related to “objects,” “situations,” “threats,” and “processes.” As illustrated in Fig. 2, taking into account recent revisions (Steinberg and Bowman 2009), the levels can be defined as follows:

- Level 0 – Sub-Object Data Assessment: estimation and prediction of signal/object observable states based on pixel/signal level data association and characterization
- Level 1 – Object Assessment: estimation and prediction of entity states based on inferences from observations

- Level 2 – Situation Assessment: estimation and prediction of entity states based on inferred relations among entities
- Level 3 – Impact Assessment: estimation and prediction of effects on situations of planned or estimated/predicted actions by the participants
- Level 4 – Process Refinement (an element of Resource Management): adaptive data acquisition and processing to support mission objectives

Low levels of JDL, namely, Levels 0 and 1, are focused in detection and tracking of in-

dividual entities, a problem usually named as multi-target tracking, while higher levels, namely, Levels 2 and 3, are related with what is happening around the entities to understand how information, events, and own actions will impact goals and objectives. Situation assessment (L2) is related with the capacity to obtain a global view of the environment, to describe the relationship between the entities tracking in the environment and to infer or describe a joint behavior. Impact assessment (L3) is related with the estimation of the impact of a special situation to the application of interest, processing and evaluating situations that are of special relevance to the scenario because they relate to some type of threatening, critical situation, or any other special world state.

Computational Resources for Big Data Fusion: Cloud Computing

Ubiquitous electronic sources such as sensors and video cast a steady stream of dynamic data, while text-based data from databases, Internet, email, social media, chat and VOIP, etc. is growing exponentially. Therefore, the high data flows, both from devices or text sources, may make systems unable to process them in real time and lead to time delays, extra costs, and even inappropriate decisions due to missing or incomplete information. Therefore, IF systems should maximize throughput while allowing complex fusion analytics to discover critical information in a huge amount of data. Hence, there is an immediate need to leverage efficient and dynamically scalable data processing infrastructure to analyze big data streams from multiple sources in a timely and scalable manner to establish accurate situation awareness (Szuster et al. 2017).

A complete ICT (information and communications Technologies) paradigm shift supports the development and delivery of IF applications depending on high-volume data streams, for instance, emergency management, to avoid getting overwhelmed by incoming data volumes, data sources, or data types. An appropriate trade-off between time and accuracy requires big data analytic architectures to support both batch data an-

alytic processes (for latent but quality solutions) and streaming data analytics (for near real-time situation awareness). This exponential explosion has a tremendous effect to the information fusion community that constantly deals with big data streams.

The deployment and provision of IF systems based on big data, like emergency management, will greatly benefit from a cloud-like middleware infrastructure, which could be formulated to create a hybrid environment consisting of multiple private (municipalities, enterprises, research organizations) and public (Amazon 2017; Microsoft 2017) infrastructure providers to deliver on-demand access to emergency management applications. Cloud computing assembles large networks of virtualized ICT services such as hardware resources (such as CPU, storage, and network), software resources (such as databases, application servers, and web servers), and applications. In industry these services are referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing services are hosted in large data centers, often referred to as data farms, operated by companies such as Amazon (A), Google Microsoft, etc. The special features of cloud, like elasticity, thin-client user interface, as well as resource pools for both data and computing, denote that the cloud technique is adequate for collecting, hosting, and processing emergency data.

However, the creation of a middleware infrastructure and an associated dynamic deployment model enabling hybrid cloud-based applications faces technical obstacles and research challenges. Cloud-based delivery of dynamic IF applications has fundamental differences from existing cloud application models (e.g., web applications, content delivery networks (CDNs), and large-scale scientific simulations), usually operated on static data in response to queries such as provision a video file (CDNs) or commit a credit card transaction (e.g., e-commerce). That is, these applications are designed to maintain efficient and fault-tolerant collection of data that is accessed and aggregated only when a query or transaction request is issued by a user. Such data manipu-

lation application model incurs high overheads when applied to IF applications that demand for high processing capacity over heterogeneous and streaming data from multiple sources with strict QoS guarantees on processing latency.

Hence, there is need to innovate a dynamically deployable, multi-cloud middleware infrastructure to establish these applications on-demand, scalable interactions with cloud resources (such as CPU, storage disks, networks, and software frameworks) to optimize quality of service (QoS) targets in terms of data analysis delay, decision-making delay, storage cost, availability, reliability, and the like. The main challenge exists in the design of intelligent technologies with integrated control criteria and mechanisms for fast access to distributed, streaming data, services, and network equipment and dynamic adaptation of the device capacities and services to traffic loads, user requirements, and scenarios of application.

Application Examples

Three representative examples have been selected to illustrate the open problems and expected IF outputs for situation assessment based on electronic devices and information sources.

(A) *Emergencies.* Natural and man-made disasters, such as tsunamis, earthquakes, floods, or epidemics, pose a significant threat to human societies. The lessons learned from the growing number of recent emergencies, such as Colorado flood (2013), Japan's earthquake (2011), Queensland flooding in Australia (2010), Haiti earthquake (2010), etc. have put impetus on the development of emergency evacuation systems. Well-coordinated emergency management activities that involves guiding of citizens out of danger areas, placement of medical team to the most appropriate locations, and real-time planning of evacuation routes before and after a disaster play a significant role in saving lives, protecting critical infrastructures, and minimizing causalities. The management of emergency activities such as guiding people

out of dangerous areas and coordinating rescue teams is dependent on the availability of historical data as well as on the effective real-time integration and utilization of data streaming from multiple sources including on-site sensors, social media feeds, and messaging on mobile devices. The ability to make sense of data by fusing it into new knowledge would provide clear advantages in emergency situations. Inadequate SA in disasters has been identified as one of the primary factors in human errors, with grave consequences such as deaths and loss of infrastructure. As an example, during the 2010 Haiti earthquake (Caragea et al. 2011), text messaging via mobile phones and Twitter made headlines as being crucial for disaster response, but only some 100,000 messages were actually processed by government agencies due to lack of automated and scalable data processing infrastructure. As large numbers of people opted for social media (and, even worse, many people re-tweeted inaccurate information), it led to a tsunami of data for agencies and rescue staff, who were not able to analyze it fast enough to respond.

(B) *Threats.* Integrating low- and high-level information may lead to significant new capabilities in the counterterrorism field (Biermann et al. 2014). In this critical area, work is currently partitioned into two main subareas: analyzing terrorism and detecting preparations for it using high-level information (intelligence reports) and immediate response to threats using sensor information. By combining the two, it will be possible to achieve shorter response times and better direction of the immediate response, enabling better preventative measures. From sensor information it is possible to detect that something is about to happen. By combining sensor data with intelligence information, it will be possible to understand more about the terrorist event and thus more easily prevent it. There are a number of practical needs from civilian, law enforcement, and security applications

for fusing information stored in databases, ontologies, or other sources. Effective use of this information potential requires tracking and data fusion systems that provide near real-time situation pictures electronically representing a complex, dynamically evolving overall picture. For example, the ability to quickly fuse information from multiple sources on organized crime or terrorist activities from the various member states and make this available to the appropriate authorities rapidly would increase safety and protect citizens. So, contextual knowledge can aid both the formulation of an improved state estimate and also be used to aid in the interpretation of a computed estimate. However, its incorporation in the fusion process adds complexity and demands that new, hybrid techniques be developed.

(C) *Healthcare.* The problem of bridging the gaps between the relatively mature technology of device-based fusion and the increasingly important, but as yet still immature, field of high-level (network or text-based) medical data remains an open-ended problem. In spite of this interest, there are, today, many ideas but no real solutions. The ability to automate the fusion of device-based and text-based information should have a huge impact on healthcare, resulting in faster, more accurate information for authorities and citizens to base their decisions upon, better logistics in the case of medical service provision and consumption and transnational medical information sharing.

Conclusion and Future Research

The synergic fusion of sensory data, natural language information, and contextual information may bring important benefits in critical problems such as protection from criminal threats, emergency/disaster situation management, border security, or healthcare. It is clear that this type of emergency situations are chaotic in nature,

and any incident management system typically encompasses multiple and diverse sources of information such as mobile devices from affected people, social network feeds, available on-site sensors, contextual information databases, etc.

Processing these big volumes of data in real time is a grand challenge in computing, considering besides that sources are available in different formats and relative qualities, scattered in different geographical areas, reliable at different confidence levels, etc. Situation awareness in big areas such as emergency scenarios becomes a complex, distributed information fusion scenario in which innovative techniques are needed to efficiently curate information sources and build inferences and assessments in real time useful for decision and sense making. Future research is required to analyze appropriate storage and computing architectures, algorithms to merge in real time the diverse data streams to meet the required performance in challenging scenarios.

References

- Amazon (2017) <https://aws.amazon.com/>. Accessed Oct 15th 2017
- Bettini C, Brdiczka O, Henricksen K, Indulska J, Nicklas D (2010) A survey of context modelling and reasoning techniques. *Pervasive Mob Comput* 6(2):161–180
- Biermann J, Garcia J, Krenc K, Nimier V, Rein K, Snidaro L (2014) Multi-level fusion of hard and soft information. 17th International Conference on Information Fusion, Salamanca. July 2014
- Biermann J, Garcia J, Krenc K, Nimier V, Rein K, Snidaro L (2016) Standardized representation via BML to support multi-level fusion of hard and soft information. Symposium IST/SET-216 Information Fusion (Hard and Soft) for Intelligence, Surveillance & Reconnaissance (ISR). Norfolk Virginia, USA 4, 5 May 2015
- Bishop C (2004) Pattern recognition and machine learning. Springer, New York
- Caragea C, McNeese N, Jaiswal A, Traylor G, Kim H-W, Mitra P, Wu D, Tapia A-H, Giles L, Jansen B-L, Yen J (2011) Classifying text messages for the Haiti earthquake. Proceedings of the 8th International ISCRAM Conference – Lisbon, Portugal, May 2011
- Gómez J, García J, Patricio MA, Molina JM, Llinas J (2011) High-level information fusion in visual sensor networks. In: Ang K-L, Seng K-P (eds) *Information processing in wireless sensor networks: technology, trends and applications*, IGI Global, Hershey, Pennsylvania

- Liggins M, Hall D, Llinas J (2008) Handbook of multi-sensor data fusion: theory and practice, 2nd edn. CRC Press, Boca Raton, Florida
- Microsoft (2017) <https://cloud.microsoft.com/en-us/>. Accessed Oct 15th 2017
- Snidaro L, Garcia J, Corchado JM (2014) Guest editorial: context-based information fusion. Inf. Fusion, Special Issue on Context-Based Information Fusion, 2014
- Snidaro L, García J, Llinas J, Blasch E (2016) Context-enhanced information fusion. Boosting real-world performance with domain knowledge. Springer, Basel, Switzerland
- Steinberg A-N, Bowman C (2009) Revisions to the JDL data fusion model, Chap. 3. In: Liggins M, Hall D, Llinas J (eds) Handbook of multisensor data fusion. CRC Press, London, pp 45–68
- Szuster P, Molina J-M, Garcia J, Kolodziej J (2017) Data fusion in cloud computing: big data approach. European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23–26, pp 569–575

Data Integration

Paolo Papotti¹ and Donatello Santoro²

¹Campus SophiaTech – Data Science Department, EURECOM, Biot, France

²Dipartimento di Matematica, Informatica ed Economia, Università degli Studi della Basilicata, Potenza, Italy

Synonyms

Information integration

Definitions

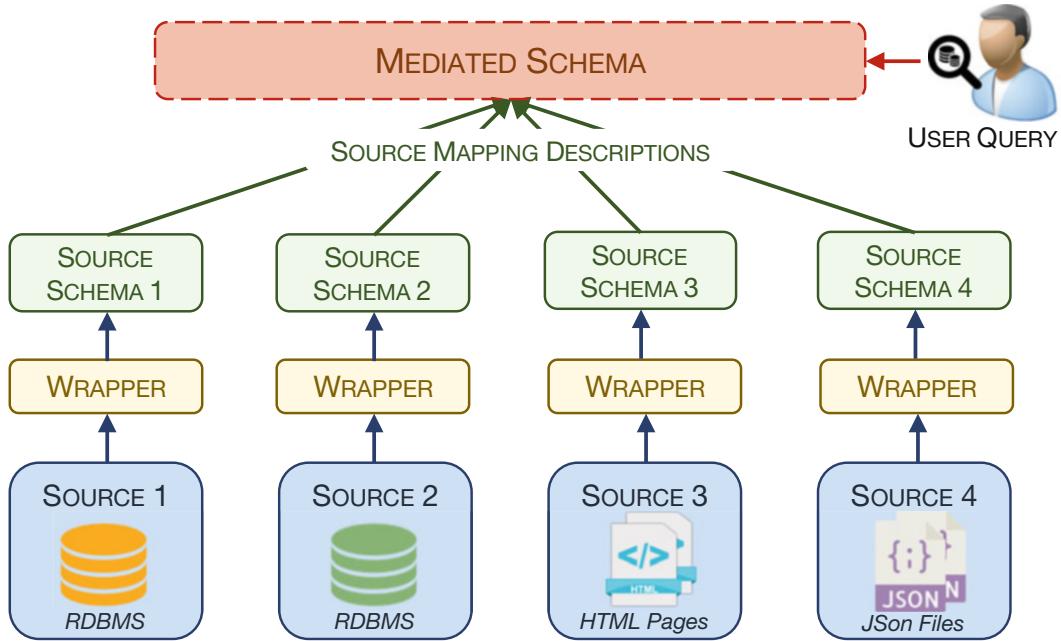
The goal of data integration systems is to provide a uniform access to a set of heterogeneous data sources. These sources can differ on the data model (relational, hierarchical, semi-structured), on the schema level, or on the query-processing capabilities. In a data integration architecture, these sources are queried by using a global schema, also called mediated schema, which provides a virtual view of the underlying sources.

Overview

Integrating data between different sources is a crucial step in many real-life applications, and the growth of structured data sources available on the Web is making this problem even more challenging. Consider as an example a Web application where users can query information about sport events planned in a particular day. In a traditional data management application, the information is stored in a database with a fixed schema (e.g., in a relational data management system) and retrieved by using a query. However, in many cases the information is stored on multiple, independent databases, which can have heterogeneous schemas and different logical organization. A data integration system solves this problem by adopting a virtual approach (Liu and Zsu 2009; Doan et al. 2012), as outlined in Fig. 1.

At the bottom of the figure, there are data sources that provide the data. Examples of these sources are HTML pages of a website, relational databases, XML or JSON files, and so on. Each source has a different schema and requires a different query. Moreover, once the results are collected from the sources, they must be merged to produce a unique answer for a given query. All these steps are not trivial and can lead to a large amount of work for users with an increasing number of sources and queries. However, with a data integration system, users do not have to execute queries directly on these sources but rather on a unified representation by using a virtual mediating schema. This schema, at the top of the figure, is defined to support the query workload and contains only the aspects of the domain that are relevant to the final application. The approach is defined *virtual* because the mediated schema does not store any data: the queries posed on the mediated schema are converted into queries on the base data sources and their results combined by the system.

To ease the extraction from the data sources, a set of wrappers are needed. The goal of each wrapper is to convert the query result of a data source into an easy to process form. For example, a wrapper for a Web form converts an input query into a set of HTTP transactions to collect results



Data Integration, Fig. 1 A typical data integration architecture

in terms of HTML pages. Finally, these pages are converted in a structured format, for example, relational tuples. To create wrappers, one can use a manual approach or rely on automatic and semi-automatic techniques to infer them (Crescenzi et al. 2001).

In order to connect the sources to the mediated schema, a set of source descriptions are defined. Essentially, these descriptions represent a semantic mapping between every source schema and the mediated schema (but not across source schemas). These mappings define correspondences between source and target attributes and take care of different structural representation across the schemas. For example, if two attributes appear in the same relation in a source and in two relations in the target schema, and the two target relations have a foreign key defined across them, then the mapping explicitly represents the semantic relationship in the source with a join between the two target relations.

A different approach for integrating data from heterogeneous sources is data warehousing. In these systems, the underlying data sources are translated and materialized into a single phys-

ical database, called warehouse. While in the virtual approach, the queries posed on the mediated schema are then translated into queries on the base sources, in a warehouse system, the queries are directly answered on the materialized database. The query execution phase is usually faster on data warehouses, but they require an additional materialization step that is expensive in terms of resources and needs to be executed periodically in order to handle new data (e.g., every night). This materialization step is usually performed by using an extract, transform, load (ETL) tool that extracts data from the sources and loads them into the warehouse. Unlike wrappers, ETL tools execute more complex transformations to the data but are procedural in nature and require IT experts to write and maintain them over time.

A data integration system I is formally defined as a triple $\langle G, S, M \rangle$ (Lenzerini 2002), where:

- G is the global (or mediated) schema;
- S is the source schema;
- M is the mapping between G and S .

Both G and S are expressed in a language over an alphabet formed by symbols for each of the relations. The definition of M is one of the crucial aspects in the design of a data integration system. In the literature two different languages have been proposed, called *local-as-view* (*LAV*) and *global-as-view* (*GAV*).

In a system based on the GAV approach, M associates each element of G with a query over S . The main idea is that each element of the global schema should be characterized in terms of a view over the sources. This approach is conceptually very easy, and it is usually used when the source schemas are stable, since adding a new source may require to refine existing mappings.

On the other hand, in a LAV-based system, the source database is modeled as a set of views over G . This approach makes the introduction of new sources easier, since defining a LAV mapping does not require knowing other data sources, but it is sufficient to enrich the mapping with new constraints, without other changes.

Example 1 Consider a data integration scenario with a schema G and four different data sources, $S1$, $S2$, $S3$, and $S4$, as follows:

G

$SportEvent(date, desc, sport, venue_id)$
 $Venue(venue_id, name, city)$

S1

$SoccerEvent(date, desc, stadium_id)$
 $Stadium(id, name, zip)$

S2

$Sport(id, name)$
 $Event(date, desc, sport_id)$

GAV Mappings

$m_1 : SportEvent(date, desc, sport, venue_id) \subseteq SoccerEvent(date, desc, venue_id)$
 $m_2 : SportEvent(date, desc, sport, venue_id) \subseteq Event(date, desc, sport_id), Sport(id, sport)$
 $m_3 : Venue(id, name, city) \subseteq Stadium(id, name, zip), ZipCodes(zip, city)$

LAV Mappings

$m_4 : FB_Event(date, desc, type, location, city) \subseteq SportEvent(date, desc, type, venue_id),$
 $Venue(venue_id, location, city)$

Data Integration, Fig. 2 LAV and GAV mappings for Example 1

S3

$FB_Event(date, desc, type, location, city)$

S4

$ZipCodes(zip, city)$

The mappings for the scenario are shown in Fig. 2. Data sources $S1$ and $S2$ can be connected to the mediated schema by using three GAV mappings. Note how the mediated relation *SportEvent* is defined as the union of two conjunctive queries, m_1 and m_2 : the first one is used to map data from a single relation *SoccerEvent*, while the second one requires a join between *Event* and *Sport*. Another important consideration is that mappings in some cases may leave some information incomplete. For example, source relation *Event* does not have data for the venue of the event, so m_2 will generate some null values. Finally m_4 is an example of a vertical partition, where a source record is divided into two sets of records in the target and is expressed as a LAV mapping. The symbol \subseteq defines the open-world assumption. In essence, the data sources are assumed to be incomplete and may not describe all the data in the domain.

Once the data integration scenario is defined, users can start asking queries over the mediated schema. Since, as discussed, this mediated schema is virtual, in order to answer user queries, the system needs to convert them into queries over the base data sources using mappings M . This process is known as *query reformulation* and generates as result of a logical query plan that needs to be optimized and executed to produce the final result. The query reformulation

D

algorithm is different in case of GAV or LAV mappings. In the former case, since the mediated schema is defined as a set of views over base relations, the reformulation algorithm is based on a simple unfolding strategy. In essence, given a query q posed over G , every relation element is substituted with the corresponding query over the sources. In case of LAV mappings, the problem of processing a query is called view-based query processing, and it is based on techniques for answering queries using views (Pottinger and Halevy 2001).

Key Research Findings

Mapping Generation

It is evident that there is no data integration without precise and complete descriptions of the semantic relationships between sources and mediated schema. Creating such *schema mappings* is an expensive process that can be achieved only by people with expertise in computer science, as mappings need to be expressed formally, and in the application domain, to understand precisely the semantics of the schemas involved.

Several systems have been proposed to assist humans in the task of generating schema mappings from high-level specifications, such as a GUI. The ultimate goal is to automate as much as possible the mapping creation, thus reducing the time and cost in this process. To support this task, several modules must be put together. Given a source and the mediated schema, the initial step is to profile them to identify possible semantic correspondences across their attributes. This task is called *schema matching* and can be done by exploiting metadata (such as attribute labels and types), overlapping data values, previous manually created matchings stored in a corpus, or combinations of such methods (Bernstein et al. 2011). After the users have validated the correspondences, the following task is to create the mappings according to the desired language. The schema mapping creation takes the correspondences and returns (possibly) complex logical formulas that can support the data integration (Popa et al. 2002).

Query Processing

As discussed in the previous sections, a query posed over the mediated schema is reformulated to produce a logical query plan that is then executed by the system over the base data sources. In order to produce an efficient query plan, a data integration system relies on optimization techniques. This step is conceptually similar to the optimization process adopted by traditional data management systems, and many techniques may be adopted from the literature. However, since a data integration system is usually virtual, dynamic, and distributed, it needs some special treatments. In general, executing a query in an integration context requires to deal with several challenges that would not arise in a traditional DBMS system. For example, in order to choose an efficient query plan, the optimization module needs to estimate the cost of the I/O operations. In a DBMS this estimation can be very accurate, while it is usually more involved in a data integration scenario. In fact, a data source may be a relational DBMS but also a REST Web Service, an HTML page, and so on. In these settings, the optimizer may have not enough information to choose the right query plan.

In addition, there are several factors that may change the performance of a query plan over time, such as varying network connectivities or new workloads for any of the sources. As a result, a plan may be good at optimization time but significantly bad in a second moment. To solve these problems, an adaptive query processing architecture has been proposed (Ives et al. 1999, 2004). The main idea is that, in a data integration context, the traditional separation between the query execution engine and the query optimizer is not longer appropriate, but there is the need of a continuous interaction between them. In particular an adaptive query processing starts with a preliminary plan, and during the query execution, this plan is updated and reevaluated, with the result that the engine may even decide to change the query plan during execution (Ives et al. 2004).

P2P Data Management

The standard data integration architecture is based on the concept of a global unified schema.

In some cases this approach will require an important effort in order to abstract the model of the underlying data sources and a collaboration between the owners of the data sources. There are contexts in which organizations want to share data, but a mediated schema approach is not feasible or too expensive. An important contribution to solve these problems is the peer-to-peer data management architecture (Halevy et al. 2003, 2006; Tatarinov et al. 2003). The main difference with a classical data integration solution is that the mediated schema is not used anymore, but rather each source (or peer) provides mappings to a subset of other peers. In the resulting P2P architecture, users pose queries with respect to a specific peer's schema. A query is then recursively reformulated and forwarded to connected peers.

Examples of Application

Data integration has several possible applications. Traditionally, it has been developed in the context of managing enterprise or institutional data repositories. The need of querying across several internal data source happens naturally in several settings, such as large projects where multiple parties interact, or large organizations with several units or departments. More recently, data integration has become more visible as a commodity with websites. Several services offer the access to multiple data sources in a unified querying interface, for example, online search for accommodation or flights.

Future Directions for Research

In the last years, the amount of data available on the Web is growing in an exponential way. For example, recent work has shown that hundreds of millions of structured Web tables can be harvested and used in applications (Balakrishnan et al. 2015; Chakrabarti et al. 2016). The traditional approach of building a uniform global schema is affordable only with a reasonable number of data sources and does not scale to these new opportunities. Instead of integrating all the

data, the challenge here is to build a system that provides a uniform search services over these data sources as automatically as possible. This is the approach proposed with dataspaces (Franklin et al. 2005), which provide base functionality over all data sources, regardless of how integrated they are. New methods and systems are needed to enable pay-as-you-go approaches to data integration, where even structured and unstructured data can be integrated and become available with little setup time (Golshan et al. 2017).

Cross-References

► Schema Mapping

References

- Balakrishnan S, Halevy AY, Harb B, Lee H, Madhavan J, Rostamizadeh A, Shen W, Wilder K, Wu F, Yu C (2015) Applying webtables in practice. In: CIDR
- Bernstein PA, Madhavan J, Rahm E (2011) Generic schema matching, ten years later. PVLDB 4(11): 695–701
- Chakrabarti K, Chaudhuri S, Chen Z, Ganjam K, He Y, Redmond W (2016) Data services leveraging bing's data assets. IEEE Data Eng Bull 39(3):15–28
- Crescenzi V, Mecca G, Merialdo P (2001) Roadrunner: towards automatic data extraction from large web sites. In: VLDB 2001, proceedings of 27th international conference on very large data bases, Roma, 11–14 Sept 2001, pp 109–118
- Doan A, Halevy A, Ives Z (2012) Principles of data integration, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco
- Franklin M, Halevy A, Maier D (2005) From databases to dataspaces: a new abstraction for information management. ACM SIGMOD Rec 34(4):27–33
- Golshan B, Halevy AY, Mihaila GA, Tan W (2017) Data integration: after the teenage years. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems, PODS 2017, Chicago, 14–19 May 2017, pp 101–106
- Halevy AY, Ives ZG, Suciu D, Tatarinov I (2003) Schema mediation in peer data management systems. In: Proceedings 19th international conference on data engineering, 2003. IEEE, pp 505–516
- Halevy A, Rajaraman A, Ordille J (2006) Data integration: the teenage years. In: Proceedings of the 32nd international conference on very large data bases, VLDB Endowment, VLDB'06, pp 9–16
- Ives ZG, Florescu D, Friedman M, Levy A, Weld DS (1999) An adaptive query execution system for data integration. In: Proceedings of the 1999 ACM SIG-

- MOD international conference on management of data, SIGMOD'99. ACM, New York, pp 299–310. <https://doi.org/10.1145/304182.304209>
- Ives ZG, Halevy AY, Weld DS (2004) Adapting to source properties in processing data integration queries. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data, SIGMOD'04, Paris, 13–18 June 2004. ACM, New York, pp 395–406. <https://doi.org/10.1145/1007568.1007613>
- Lenzerini M (2002) Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS'02. ACM, New York, pp 233–246. <https://doi.org/10.1145/543613.543644>
- Liu L, Zsu MT (2009) Encyclopedia of database systems, 1st edn. Springer, Incorporated, New York/London
- Popa L, Velegrakis Y, Miller RJ, Hernández MA, Fagin R (2002) Translating web data. In: Proceedings of 28th international conference on very large data bases, VLDB 2002, Hong Kong, 20–23 Aug 2002, pp 598–609
- Pottinger R, Halevy A (2001) Minicon: a scalable algorithm for answering queries using views. VLDB J 10(2–3):182–198
- Tatarinov I, Ives Z, Madhavan J, Halevy A, Suciu D, Dalvi N, Dong XL, Kadiyska Y, Miklau G, Mork P (2003) The piazza peer data management project. ACM SIGMOD Rec 32(3):47–52

Data Lake

Christoph Quix¹ and Rihan Hai²

¹Fraunhofer-Institute for Applied Information Technology FIT, Sankt Augustin, Germany

²RWTH Aachen University, Aachen, Germany

Definitions

A data lake is a data repository in which datasets from multiple sources are stored in their original structures. It should provide functions to extract data and metadata from heterogeneous sources and to ingest them into a hybrid storage system. In addition, a data lake should offer a data transformation engine, in which datasets can be transformed, cleaned, and integrated with other datasets. Finally, interfaces to explore and to query the data and metadata of a data lake should be also available in a data lake system.

Overview

The term “data lake” (DL) was first mentioned by James Dixon in 2010 in a blog post (<https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/>) where he put data marts on the same level as bottled water, which is cleansed, packaged, and structured for easy consumption. In contrast, a data lake manages the raw data as it is ingested from the data sources.

In the initial article (and in a later, more detailed article (<https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/>) in 2014), DLs are described as systems that store data from a *single* source. However, most people consider a DL today as a general enterprise-wide data repository which gets data from multiple sources. Still, it is important that there is some governance about the ingestion process for a DL; a DL is *able to* take data from any kind of data source, but in order to avoid a data swamp, the ingested data should fulfill some minimal requirements (e.g., providing metadata and a description of the raw data format). Gartner pointed this out also in their criticism (<http://www.gartner.com/newsroom/id/2809117>) of the DL concept. Especially, a proper metadata management is frequently mentioned as a basic requirement for a DL system, but details about the required metadata services (e.g., model or functions) are often missing.

Key Research Findings

Architecture

Some architecture models have been proposed, some are abstract sketches from industry (e.g., by pwc in Stein and Morrison (2014) or by podium data (<http://www.podiumdata.com/solutions/>)), and some are more concrete from research prototypes (e.g., Terrizzano et al. 2015). These proposals usually describe conceptual architectures, functional requirements, and possible product components, but they do not provide the theoretical or implementation details that would enable repeatable implementations, as these scenario-specific solutions may not apply

to a wider usage of DLs (Douglas and Curino 2015).

As Hadoop is also able to handle any kind of data in its distributed file system, many people think that “Hadoop” is the complete answer to the question on how a DL should be implemented. Of course, Hadoop is good at managing the huge amount of data with its distributed and scalable file system, but it does not provide detailed metadata functionalities which are required for a DL. For example, the DL architecture presented in Boci and Thistlethwaite (2015) shows that a DL system is a complex ecosystem of several components and that Hadoop provides only a part of the required functionality.

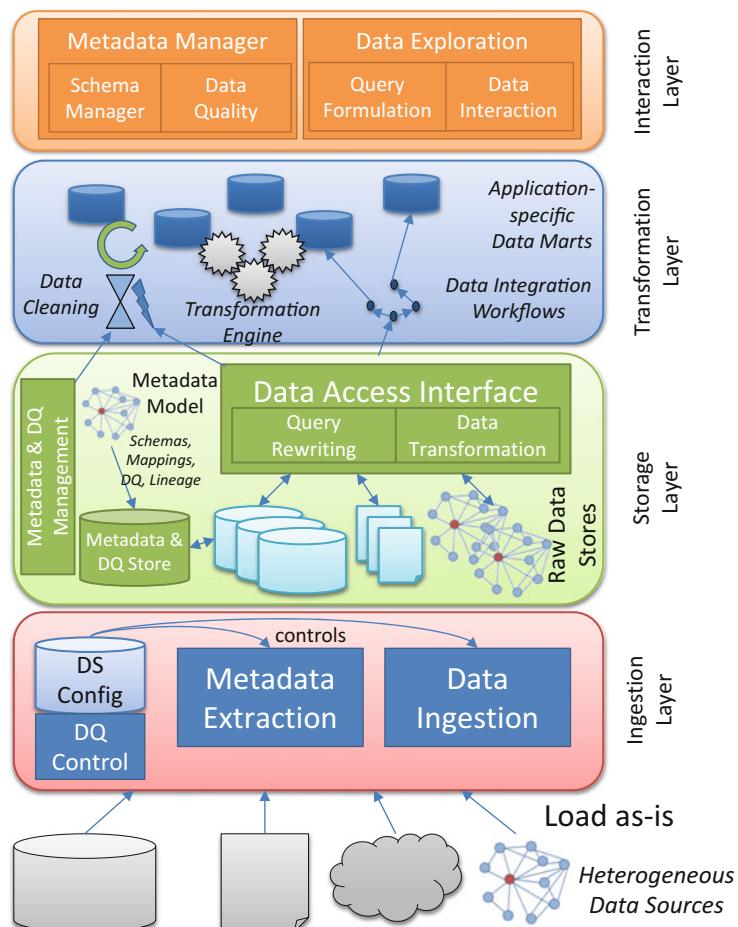
More recent articles about data lakes (e.g., Mathis 2017; LaPlante and Sharma 2016) mention common functional components of a data

lake architecture for ingesting, storing, transforming, and using data. These components are sketched in the architecture of Fig. 1 (Jarke and Quix 2017).

The architecture is separated into four layers: the *ingestion layer*, the *storage layer*, the *transformation layer*, and the *interaction layer*.

D

Data Lake, Fig. 1 Data lake architecture



information about the data sources, illustrated in the figure as *DS Config*. The metadata extractor needs to be able to detect schemas in semi-structured data (e.g., JSON or XML sources). The extracted metadata is managed by a metadata repository in the storage layer.

In addition to the metadata, the raw data also needs to be ingested into the DL. According to the idea, that the raw data is kept in its original format, this is more like a “copy” operation and thereby certainly less complex than an ETL process in data warehouses. Nevertheless, the data needs to be put into the storage layer of the DL, which might imply some syntactical transformation or loading of the data into the “raw data stores” in the storage layer. The loading process can be done in a lazy way, i.e., only if a user requests data from this particular data source (Karæz et al. 2013).

Data governance and data quality (DQ) management are important in DLs to avoid data swamps. The *DQ control* component should make sure that the ingested data has a minimal quality. For example, if a source with information about genes is considered, it should also provide an identifier of the genes in one of the common formats (e.g., from the Gene Ontology) instead of using proprietary IDs that cannot be mapped to other sources. Due to the huge volume and variety of the data, it is probably not be possible to specify all DQ rules manually; here, also automatic support is required to detect such rules and then also to evaluate these rules in a fuzzy way (Saha and Srivastava 2014). Also, data profiling techniques can help to identify patterns in the source data (Abedjan et al. 2015).

Storage Layer

The main components in the storage layer are the metadata repository and the repositories for raw data. The *metadata repository* stores all the metadata of the DL which have been partially collected automatically in the ingestion layer or will be later added manually during the curation or usage of the DL. The metadata should also include feedback from the users of the DL, e.g., which attributes have been used to join datasets, what kind of transformations (for integration or

data cleaning) have been applied to datasets, or for which analytical reports a dataset has been relevant. This information will be useful for subsequent usage of the datasets as it contains knowledge on how the dataset can be used. The main challenge for the metadata repository is the definition of a *metadata model*, which, on the one hand, is generic enough to represent the wide variety of metadata in a DL, and on the other hand, has detailed and concrete semantics for the metadata items. Furthermore, it should have a manageable complexity such that the metadata can be also exploited by the end users.

The raw data repositories are the core of the DL in terms of data volume. As the ingestion layer provides the data in its original format, different storage systems for relational, graph, XML, or JSON data have to be provided. Moreover, the storage of files using proprietary formats should be supported. Hadoop seems to be a good candidate as a basic platform for the storage layer, but it needs to be complemented with components to support the data fidelity, such as Apache Spark (<https://spark.apache.org/>). In order to provide a uniform way to query and access the data to the user, the hybrid data storage infrastructure should be hidden by a uniform *data access interface*. This data access interface should provide a query language and a data model, which have sufficient expressive power to enable complex queries and represent the complex data structures that are managed by the DL. Current systems such as Apache Spark and HBase (<https://hbase.apache.org/>) offer this kind of functionality using a variant of SQL as query language and data model. Another option is to use JSON as unifying data representation as this is supported by many NoSQL systems. Still, a problem is the lack of a standardized query language for JSON, but proposals as JSONiq (Florescu and Fourny 2013), which are based syntactically and semantically on the well-known query languages SQL and XQuery, are promising directions.

A major challenge is the rewriting of the user queries into the specific query languages of the raw data repositories. The classical query rewriting problem has been addressed in data integration for many years in which the main

focus was correctness and completeness of the rewritten query. In addition to this problem, many other aspects have to be considered for the rewriting in the DL system. For example, the costs for data transformation between the different data formats have also to be taken into account (When relational and JSON data should be joined, is it more efficient to translate the JSON data into a relational form or vice versa?) or the costs for moving the data between different nodes of the distributed storage system.

Transformation Layer

To transform the raw data into a desired target structure, the DL needs to offer a data transformation engine in which operations for data cleaning, data transformation, and data integration can be realized in a scalable way. In contrast to a data warehouse, which aims at providing one integrated schema for the all data sources, a DL should support the ability to create *application-specific* data marts, which integrate a *subset* of the raw data in the storage layer for a concrete application. From the logical point of view, these data marts are rather a part the interaction layer, as the data marts will be created by the users during the interaction with the DL. On the other hand, their data will be stored in one of the systems of the storage layer. The idea of data marts in this context is similar to the concept of personal information spaces, which were initially proposed in the context of dataspaces (Franklin et al. 2005), and support the idea of “pay-as-you-go” integration (Jeffery et al. 2008). The knowledge created during the definition of the data mart (e.g., information about how to transform, integrate, and analyze datasets) needs to be maintained at the transformation layer and should be captured in the metadata store.

In addition, data marts can be more application-independent if they contain a general-purpose dataset which has been defined by a data scientist. Such a dataset might be useful in many information requests of the users.

Interaction Layer

The top layer focuses at the interaction of the users with the DL. The users will have to access

the metadata to see what kind of data is available and can then explore the data. Thus, there needs be a close relationship between the *data exploration* and the *metadata manager* components. On the other hand, the metadata generated during data exploration (e.g., semantic annotations, discovered relationships) should be inserted into the metadata store using the *schema manager* and *enrichment* modules.

The *query formulation* component should support the user in creating formal queries that express her information requirement. As stated earlier, we cannot expect that the user is able to handle the functions offered by the data access system directly. The *data interaction* should cover all the functionalities which are required to work with the data, including visualization, annotation, selection and filtering of data, and basic analytical methods. More complex analytics involving machine learning and data mining is in our vision not part of the core of a DL system.

Lazy and Pay-as-You-Go Concepts

A basic idea of the DL concept is to consume as little upfront effort as possible and to spend additional work during the interaction with users, e.g., schemas, mappings, and indexes are created while the users are working with the DL. This has been referred to as lazy (e.g., in the context of loading database (Karæz et al. 2013)) and pay-as-you-go techniques (e.g., in data integration Sarma et al. 2008). All workflows in a DL system have to be verified, whether the deferred or incremental computation of the results is applicable in that context. For example, metadata extraction can be done first in a shallow manner by extracting only the basic metadata; only detailed data of the source is required; a more detailed extraction method will be applied.

A challenge for the application of these “pay-as-you-go” methods is to make them really “incremental,” e.g., the system must be able to detect the changes and avoid a complete recomputation of the derived elements. This challenge is also addressed in a Lambda architecture in which the speed layer should provide the incremental updates to the results computed by the batch layer.

Schema-on-Read and Evolution

DLS also provide access to un- or semi-structured data for which a schema was not explicitly given during the ingestion phase. Schema-on-read means that schemas are only created when the data is accessed, which is in-line with the “lazy” concept described in the previous section. The shallow extraction of metadata might also lead changes at the metadata level, as schemas are being refined if more details of the data sources are known. Also, if a new data source is added to the DL system, or an existing one is updated, some integrated schemas might have to be updated as well, which leads to the problem of schema evolution (Curino et al. 2013).

Another challenge to be addressed for schema evolution is the heterogeneity of the schemas and the frequency of the changes. While data warehouses have a relational schema which is usually not updated very often, DLs are more agile systems in which data and metadata can be updated very frequently. The existing methods for schema evolution have to be adapted to deal with the frequency and heterogeneity of schema changes in a big data environment (Hartung et al. 2011).

Mapping Management

Mapping management is closely related to the schema evolution challenge. Mappings state how data should be processed on the transformation layer, i.e., to transform the data from its raw format as provided by the storage layer to a target data structure for a specific information requirement. The definition of mapping languages and operations on mappings (e.g., composition) has been studied in the field of model management (Bernstein and Melnik 2007), an earlier research area at a time in which schema-less systems were not yet relevant and models were seen as an important asset in the design of interoperable information systems. Although heterogeneity of data and models has been considered and generic languages for models and mappings have been proposed (Kensche et al. 2009), the definition and creation of mappings in a schema-less world has not received much attention, yet. There has been some work in the context

of regular path queries for graph databases; for example, (Calvanese et al. 2012) defined the term “relative containment” for queries wrt. the sources that are available. Such a query language could be also applicable for a heterogeneous DL system.

A prerequisite for the definition of formal mappings is the identification of correspondences or similarities. If schemas are given explicitly, then schema matching approaches can support this task (Hartung et al. 2011); however, in DLs the raw data is less structured, and schema information is not explicitly available. Thus, in these contexts methods for data profiling or data wrangling have to be combined with schema extraction and relatable dataset discovery (Alserafi et al. 2017).

Query Rewriting and Optimization

There is a trade-off between expressive power and complexity of the rewriting procedure as the complexity of query rewriting depends to a large degree on the choice for the mapping and query language. However, query rewriting should not only consider completeness and correctness, but also the costs for executing the rewritten query should be taken into account. Thus, the methods for query rewriting and query optimization require a tighter integration (Gottlob et al. 2014).

It is also an open question whether there is a need for an intermediate language in which data and queries are translated to do the integrated query evaluation over the heterogeneous storage system or whether it is more efficient to use some of the existing data representations. Furthermore, given the growing adoption of declarative languages in big data systems, query processing and optimization techniques from the classical database systems could be applied as well in DL systems.

Data Governance and Data Quality

Since the output of a DL should be useful knowledge for the users, it is important to prevent a DL from becoming a *data swamp*. Similar with data warehouses, incomplete, inaccurate, and ambiguous data would jeopardize the query result accuracy, even worse, lead to unusable result

(Jarke et al. 1999). There is a trade-off between data openness and quality control. It is often mentioned that data governance is required for a DL. First of all, data governance is an organizational challenge, i.e., roles have to be identified, stakeholders have to be assigned to roles and responsibilities, and business processes need to be established to organize various aspects around data governance (Otto 2011). Still, data governance needs to be also supported by appropriate techniques and tools. For data quality, as one aspect of data governance, a similar evolution has taken place; the initially abstract methodologies have been complemented in the meantime by specific techniques and tools. Preventive, process-oriented data quality management (in contrast to data cleaning, which is a reactive data quality management) also addresses responsibilities and processes in which data is created in order to achieve a long-term improvement of data quality.

Applications of Data Lakes

Although data lakes are a young concept, many organizations are investigating or investing in data lake solutions. Organizations with a good data management architecture, established data governance, and several data integration solutions already in place (e.g., data warehouses) are considering data lakes as a complementary solution to their existing systems. However, the added value of a data lake in such situations might be low as many of the benefits of data lakes might be already covered by some of the existing systems.

In cases where the data management of an organization is much less developed, a data lake might be an approach to start data governance initiatives and to think about the business processes in a data-oriented way (e.g., where is data created, where is it consumed). Many organizations have a scattered data management landscape with hundreds or even thousands of partially interconnected, partially isolated databases. Especially in the case of data silos which are difficult to access, data lakes might provide a significant benefit as it

removes the hurdles to locate and to get physical access to a data source.

This situation can be often seen in industrial companies in which data has not been considered as a valuable resource. Now, with more Industry 4.0 applications and success stories becoming available, companies are examining data lakes as a potential solution to manage the data produced by their production processes or by their products. Also in the research and development department (industrial or academic), data lakes can be successful because of the heterogeneity of the sources and required flexibility in data analysis (Quix et al. 2016). A slightly different concept is the idea of a virtual or logical data lake, i.e., only metadata will be extracted from the sources and made available in a unifying metadata repository. For example, Google implements this idea in their GOODS system (Halevy et al. 2016). Such metadata registries with an overview of the datasets that are managed by an organization are also important in the context of the data privacy laws established in Europe, e.g., organizations must be able to report about the datasets with personal information managed by them.

Future Directions for Research

The approach of a raw data store in which data is only loosely connected is appealing to solve problems in getting access to the relevant data that is available in an organization. Yet, the idea of data lakes is still relatively new and has not seen much attention in research. Commercial data integration systems have addressed the data lake idea and are marketing their systems as solutions to the challenges to be addressed in data lake systems, e.g., methods for metadata extraction and metadata management. Still, there are many opportunities for research, starting from defining more general reference architectures to identify the important components of a data lake, to developing new methods for specific problems in the data lake context (e.g., schema summarization, identifying relatable datasets, data governance, query processing on heterogeneous datasets).

Cross-References

- ▶ [Big Data Indexing](#)
- ▶ [Data Fusion](#)
- ▶ [Data Integration](#)
- ▶ [Data Profiling](#)
- ▶ [Data Wrangling](#)
- ▶ [ETL](#)
- ▶ [Schema Mapping](#)
- ▶ [Uncertain Schema Matching](#)

References

- Abedjan Z, Golab L, Naumann F (2015) Profiling relational data: a survey. VLDB J 24(4):557–581. <https://doi.org/10.1007/s00778-015-0389-y>
- Alserafi A, Calders T, Abelló A, Romero O (2017) Dsprox: dataset proximity mining for governing the data lake. In: Beecks C, Borutta F, Kröger P, Seidl T (eds) Proceedings of 10th international conference similarity search and applications, SISAP 2017, Munich, 4–6 Oct 2017. Lecture notes in computer science, vol 10609, pp 284–299. Springer. https://doi.org/10.1007/978-3-319-68474-1_20
- Bernstein PA, Melnik S (2007) Model management 2.0: manipulating richer mappings. In: Zhou L, Ling TW, Ooi BC (eds) Proceedings of ACM SIGMOD international conference on management of data. ACM Press, Beijing, pp 1–12. <https://doi.org/10.1145/1247480.1247482>
- Boci E, Thistletonwaite S (2015) A novel big data architecture in support of ads-b data analytic. In: Proceedings of integrated communication, navigation, and surveillance conference (ICNS), pp C1-1–C1-8. <https://doi.org/10.1109/ICNSURV.2015.7121218>
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2012) Query processing under glav mappings for relational and graph databases. Proc VLDB Endow 6(2):61–72
- Curino C, Moon HJ, Deutsch A, Zaniolo C (2013) Automating the database schema evolution process. VLDB J 22(1):73–98
- Douglas C, Curino C (2015) Blind men and an elephant coalescing open-source, academic, and industrial perspectives on bigdata. In: Gehrke J, Lehner W, Shim K, Cha SK, Lohman GM (eds) 31st IEEE international conference on data engineering, ICDE 2015, Seoul, 13–17 Apr 2015. IEEE Computer Society, pp 1523–1526. <https://doi.org/10.1109/ICDE.2015.7113417>. <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7109453>
- Florescu D, Fourny G (2013) Jsoniq: the history of a query language. IEEE Internet Comput 17(5):86–90
- Franklin M, Halevy A, Maier D (2005) From databases to dataspaces: a new abstraction for information management. SIGMOD Rec 34(4):27–33. <https://doi.org/10.1145/1107499.1107502>
- Gottlob G, Orsi G, Pieris A (2014) Query rewriting and optimization for ontological databases. ACM Trans Database Syst 39(3):25:1–25:46. <https://doi.org/10.1145/2638546>
- Halevy AY, Korn F, Noy NF, Olston C, Polyzotis N, Roy S, Whang SE (2016) Managing Google’s data lake: an overview of the goods system. IEEE Data Eng Bull 39(3):5–14. <http://sites.computer.org/debull/A16sept/p5.pdf>
- Hartung M, Terwilliger JF, Rahm E (2011) Recent advances in schema and ontology evolution. In: Bellahsene Z, Bonifati A, Rahm E (eds) Schema matching and mapping, data-centric systems and applications. Springer, pp 149–190. <https://doi.org/10.1007/978-3-642-16518-4>
- Jarke M, Quix C (2017) On warehouses, lakes, and spaces: the changing role of conceptual modeling for data integration. In: Cabot J, Gómez C, Pastor O, Sancho M, Teniente E (eds) Conceptual modeling perspectives. Springer, pp 231–245. https://doi.org/10.1007/978-3-319-67271-7_16
- Jarke M, Jeusfeld MA, Quix C, Vassiliadis P (1999) Architecture and quality in data warehouses: an extended repository approach. Inf Syst 24(3):229–253
- Jeffery SR, Franklin MJ, Halevy AY (2008) Pay-as-you-go user feedback for dataspace systems. In: Wang JTL (ed) Proceedings of ACM SIGMOD international conference on management of data. ACM Press, Vancouver, pp 847–860. <https://doi.org/10.1145/1376616.1376701>
- Karæz Y, Ivanova M, Zhang Y, Manegold S, Kersten ML (2013) Lazy ETL in action: ETL technology dates scientific data. PVLDB 6(12):1286–1289. <http://www.vldb.org/pvldb/vol6/p1286-kargin.pdf>
- Kensche D, Quix C, Li X, Li Y, Jarke M (2009) Generic schema mappings for composition and query answering. Data Knowl Eng 68(7):599–621. <https://doi.org/10.1016/j.datak.2009.02.006>
- LaPlante A, Sharma B (2016) Architecting data lakes. O'Reilly Media, Sebastopol, CA, USA
- Mathis C (2017) Data lakes. Datenbank-Spektrum 17(3):289–293. <https://doi.org/10.1007/s13222-017-0272-7>
- Otto B (2011) Data governance. Bus Inf Syst Eng 3(4):241–244. <https://doi.org/10.1007/s12599-011-0162-8>
- Quix C, Berlage T, Jarke M (2016) Interactive pay-as-you-go-integration of life science data: the HUMIT approach. ERCIM News 2016(104). <http://ercim-news.ercim.eu/en104/special/interactive-pay-as-you-go-integration-of-life-science-data-the-humit-approach>
- Saha B, Srivastava D (2014) Data quality: the other face of big data. In: Cruz IF, Ferrari E, Tao Y, Bertino E, Trajcevski G (eds) Proceedings of 30th international conference on data engineering (ICDE). IEEE,

- Chicago, pp 1294–1297. <https://doi.org/10.1109/ICDE.2014.6816764>
- Sarma AD, Dong X, Halevy AY (2008) Bootstrapping pay-as-you-go data integration systems. In: Wang JTL (ed) Proceedings of ACM SIGMOD international conference on management of data. ACM Press, Vancouver, pp 861–874
- Stein B, Morrison A (2014) The enterprise data lake: better integration and deeper analytics. <http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/assets/pdf/pwc-technology-forecast-data-lakes.pdf>
- Terrizzano I, Schwarz PM, Roth M, Colino JE (2015) Data wrangling: the challenging journey from the wild to the lake. In: 7th Biennial conference on innovative data systems (CIDR). http://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper2.pdf

Data Longevity and Compatibility

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

Data archives; Data decay; Media lifespan; Storage failure

Definition of Entry

Whether stored locally or in the cloud, data ultimately resides on physical media that are subject to electrical and physical deterioration and format obsolescence, making it necessary to augment the physical storage of data with a logical organization to protect data integrity and ensure its longevity.

Overview

Like many other attributes of computing and digital systems, the volume of data produced in the world is rising exponentially (Denning and Lewis 2016; Hilbert and Gomez 2011), with a

D

growth rate that is even higher than those of circuit density and processor performance, modeled by Moore's law (Brock and Moore 2006). A few exabytes of data generation per day in the early 2010s is slated to rise to many yottabytes in the 2020s (Cisco Systems 2017; Jacobson 2013). As data gains ever-greater value in the operation of social and business enterprises, data management, integrity, and preservation become major concerns. Any physical storage medium is subject to decay over time and has a finite lifespan. Some of these lifespans are relatively short compared with desirable data retention periods in practical settings. The format in which data is stored tends to become obsolete as well. It follows that an active strategy for ensuring the integrity and longevity of stored data is an important part of any data management plan.

Media for Long-Term Data Storage

The media used for storing data have undergone significant changes over time (Goda and Kitsuregawa 2012). Data resources are maintained in hierarchical arrangements, with hardware components in the hierarchy ranging from superfast (but expensive and, thus, low-capacity) caches near processing resources to vast (but relatively slow) data vaults with ultralow storage costs. The goal of the management scheme for such hierarchical arrangements is to make the data items of highest current value reside in faster storage, where they can be efficiently accessed, with movements between the levels orchestrated so as to be responsive to changes in data values. Devices used for storing data for immediate access are known as hot storage media, in contrast to cold media that store data archives that are not in current use.

The order-of-magnitude correspondences shown in Table 1 are helpful in putting data volumes in perspective. In rough terms, the first four rows of the table (up to TBs) are currently within the domain of personal storage (although the numbers will no doubt expand in the coming years), whereas the last four are within the

Data Longevity and Compatibility, Table 1 Developing an appreciation for data volumes

Abbr.	Name	Bytes	Example(s)
KB	Kilobyte	10^3	One page of text or a small contact list
MB	Megabyte	10^6	One photo or a short YouTube video
GB	Gigabyte	10^9	A movie
TB	Terabyte	10^{12}	Netflix's movies, 4 years worth of watching
PB	Petabyte	10^{15}	Data held by an e-commerce site or bank
EB	Exabyte	10^{18}	Google's data centers
ZB	Zettabyte	10^{21}	WWW size or capacity of all hard drives
YB	Yottabyte	10^{24}	Worldwide daily data production by the 2020s

purview of large organizations, municipalities, or nation-states. At the rate data production is growing, it won't be long before we will have to decide whether to adopt proposed prefixes for 10^{27} and 10^{30} (Googology Wiki 2018).

Data lifetimes range from seconds (for temporary results kept in scratchpad or working memory, as database transactions run to completion) to centuries or more (for historical archives). Each combination of data lifetime duration and data access requirement dictates the use of certain storage device types. Setting volatile semiconductor memories aside because of their nonpermanence, the most common storage technologies used in the recent past are listed in Table 2.

Data Decay and Device Lifespans

All storage media decay over time, although the degradation mechanism, and thus methods for dealing with it, is technology-dependent. Degradations that are small-scale and/or local can be compensated for through error-correcting codes. However, once a certain number of data errors have been corrected automatically, the storage medium must be discarded and a fresh copy of the data created in a different place, because continued deterioration will eventually exceed the codes' error-correcting capacity, leading to data loss.

Magnetic storage media, the most commonly used current technology for large-scale and long-term data repositories (hot or cold), degrade through weakening or loss of magnetization, a process that is accelerated by external magnetic

fields, heat, vibration, and a number of other environmental factors. Generally, the result of such deterioration is partial data erasure, rather than arbitrary changes to the stored information. Thus, when codes are used, they can be of the erasure variety (Plank 2013; Rizzo 1997), which imply lower redundancy compared with codes for correcting arbitrary errors. Archival magnetic media, such as reel tapes, must be stored under carefully regulated environmental conditions to maximize their lifespans.

Optical storage media can decay due to the breakdown of material onto which data is stored. Here too storing the media under proper environmental conditions can reduce data decay and increase the media's lifespan. When longer lifespans are desired, storage media of higher quality must be procured. In particular, M-discs and other specially developed archival media can prolong lifespans to many centuries (Jacobi 2018; Lunt et al. 2013). Such long lifespans can be validated only via accelerated testing (Svrcek 2009), a process whose results are not as trustworthy as those obtained through direct testing in actual field use.

Solid-state media use electrical charges to store data. Imperfect insulation can lead to charges leaking out and thus data being lost. Given that the device itself degrades much more slowly, refreshing data before total decay is one possible strategy for avoiding data loss. This process is in effect very similar to refreshing in DRAM chips (Bhati et al. 2016), except that the refreshing occurs at much slower rates and, thus, its performance hit and power waste are not serious issues.

Data Longevity and Compatibility, Table 2 Attributes of storage devices in use over the past few decades

Type	Capacity	Cost	Speed	Life (years)	Archival	Additional information
Floppy disk	MB	Low	Low	5–7	No	Obsolete technology/format
CD/DVD	GB	Low	Medium	3–10	Yes	Are becoming less prominent
Blu-ray	GB+	Low	Medium	50	Yes	Still used for small-scale archives
M-disc	GB+	Low	Medium	1000	Yes	Lifespan unverified
Memory card	GB	High	High	5–10	No	Temporary personal storage
Cassette tape	GB	Low	Low	10–20	No	Obsolete technology/format
Cartridge	GB+	Low	Low	10–20	Yes	Used in mechanically accessed vaults
Hard disk	TB	Medium	Medium	3–5	No	Increasingly being replaced by SSDs
SSD (flash)	TB	High	High	5–10	No	Lifespan varies with write frequency
Reel tape	TB+	Low	Low	20–30	Yes	Lifespan with proper storage, low use
Disk array	PB	Medium	Medium	10–20	Yes	Lifespan improved by fault tolerance

Sources differ widely on the useful lifespans of data storage media. This is in part due to the fact that manufacturing quality associated with various suppliers and environmental conditions under which the media operate varies widely. Some typical figures for the most common media are included in Table 2. These figures should be used with caution, as there is no guarantee that even the lower ends of the cited ranges will hold in practice.

Preventing, and Recovering from, Data Decay

Countermeasures for avoiding data decay are very similar in nature to those used to deal with data corruption, loss, or annihilation, discussed in this encyclopedia's entry “Data Replication and Encoding” and in general references on dependable and fault-tolerant computing (Parhami 2018). Low redundancy coding can counteract small-scale or local damages (Rao and Fujiwara 1989), while replication helps prevent widespread damage. Of course, replication comes with a high storage cost, which will be even more prohibitive in the age of big data. For this reason, data dispersion (Iyengar et al. 1998; Rabin 1989) and network coding (Dimakis et al. 2011) schemes, which in effect combine the advantages of low redundancy coding with the error correction strengths of replication, will

be the preferred methods for large collections of data.

Decades of experience with building and managing large database systems are being applied to the design of big data repositories that are nonvolatile and long life (Arulraj and Pavlo 2017). However, new challenges arising from enormous amounts of less structured and unstructured data remain to be overcome. Recovery methods (Arulraj et al. 2015) are both device- and application-dependent, so they need continual scrutiny and updating as storage devices evolve, and data is used in previously unimagined volumes and ways.

Future Directions

The importance of building large-scale, nonvolatile data archives in order to safeguard immense volumes of valuable data has been recognized, to the extent of being discussed in the mass media (Coughlin 2014). Many projects are trying to achieve this goal (Yan et al. 2017). Some of the components needed for such system are already available and others are being invented.

On the storage technology side, disk arrays (Chen et al. 1994), which have played an important role in satisfying our data storage capacity and reliability needs for many years, must be adapted to the challenges presented by big data. We currently don't view semiconductor memories as suitable for data archives, but this

assessment may change in light of technology advances (Qian et al. 2015).

On the data structuring, management, and maintenance side, data centers and their interconnection networks will play a key role in the development of robust permanent data archives (Chen et al. 2016). Improved understanding of storage device failure mechanisms and data decay (Petyscale Data Storage Institute 2012; Schroeder and Gibson 2007) is another line that should be pursued.

Finally, development of new technologies for data storage, discussed in this encyclopedia under “Storage Technologies for Big Data” and “Emerging Hardware Technologies,” will no doubt be accelerated as the storage and longevity needs to expand. Examples abound, but two promising avenues currently being pursued are storing vast amounts of data on DNA (Bornholt et al. 2016; Goldman et al. 2013) and building memories based on emerging nanotechnologies involving memristors and other new device types (Menon and Gupta 1999; Strukov et al. 2008).

Cross-References

- ▶ [Data Replication and Encoding](#)
- ▶ [Storage Hierarchies for Big Data](#)
- ▶ [Storage Technologies for Big Data](#)
- ▶ [Structures for Large Data Sets](#)

References

- Arulraj J, Pavlo A (2017) How to build a non-volatile memory database management system. In: Proceedings of the ACM international conference on management of data, Chicago, pp 1753–1758
- Arulraj J, Pavlo A, Dulloor SR (2015) Let's talk about storage & recovery methods for non-volatile memory database System. In: Proceedings of the ACM international conference on management of data, Melbourne, pp 707–722
- Bhati I, Chang M-T, Chishti Z, Lu S-L, Jacob B (2016) DRAM refresh mechanisms, penalties, and trade-offs. *IEEE Trans Comput* 65(1):108–121
- Bornholt J, Lopez R, Carmean DM, Ceze L, Seelig G, Strauss K (2016) A DNA-based archival storage system. *ACM SIGOPS Oper Syst Rev* 50(2):637–649
- Brock DC, Moore GE (eds) (2006) Understanding Moore's law: four decades of innovation. Chemical Heritage Foundation, Philadelphia
- Chen PM, Lee EK, Gibson GA, Katz RH, Patterson DA (1994) RAID: high-performance reliable secondary storage. *ACM Comput Surv* 26(2):145–185
- Chen T, Gao X, Chen G (2016) The features, hardware, and architectures of data center networks: a survey. *J Parallel Distrib Comput* 96:45–74
- Cisco Systems (2017) The zettabyte era: trends and analysis. White paper. <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>
- Coughlin T (2014) Keeping data for a long time, Forbes, available on-line at <http://www.forbes.com/sites/tomcoughlin/2014/06/29/keeping-data-for-a-long-time/#aac168815e26>
- Denning PJ, Lewis TG (2016) Exponential Laws of computing growth. *Commun ACM* 60(1):54–65
- Dimakis AG, Ramachandran K, Wu Y, Suh C (2011) A survey on network codes for distributed storage. *Proc IEEE* 99(3):476–489
- Goda K, Kitsuregawa M (2012) The history of storage systems. *Proc IEEE* 100:1433–1440
- Goldman N, Bertone P, Chen S, Dessimoz C, LeProust EM, Sipos B, Birney E (2013) Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* 494(7435):77–80
- Googology Wiki (2018) SI prefix. On-line document. http://googology.wikia.com/wiki/SI_prefix. Accessed 23 Feb 2018
- Hilbert M, Gomez P (2011) The World's technological capacity to store, communicate, and compute information. *Science* 332:60–65
- Iyengar A, Cahn R, Garay JA, Jutla C (1998) Design and implementation of a secure distributed data repository. IBM Thomas J. Watson Research Division, Yorktown Heights
- Jacobi J (2018) M-Disc optical media reviewed: your data, good for a thousand years. PCWorld. On-line document. <http://www.pcworld.com/article/2933478/storage/m-disc-optical-media-reviewed-your-data-good-for-a-thousand-years.html>
- Jacobson R (2013) 2.5 quintillion bytes of data created every day: how does CPG & retail manage it? IBM Industry Insights. <http://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/>
- Lunt BM, Linford MR, Davis RC, Jamieson S, Pearson A, Wang H (2013) Toward permanence in digital data storage. *Proc Arch Conf* 1:132–136
- Menon AK, Gupta BK (1999) Nanotechnology: a data storage perspective. *Nanostruct Mater* 11(8):965–986
- Parhami B (2018) Dependable computing: a multi-level approach, draft of book manuscript, available on-line at http://www.ece.ucsb.edu/~parhami/text_dep_comp.htm
- Petyscale Data Storage Institute (2012) Analyzing failure data. Project Web site: <http://www.pdl.cmu.edu/PDSI/FailureData/index.html>

- Plank JS (2013) Erasure codes for storage stems: a brief primer. Usenix Mag 38(6):44–50
- Qian C, Huang L, Xie P, Xiao N, Wang Z (2015) Efficient data management on 3D stacked memory for big data applications. In: Proceedings of the 10th international design & test symposium, Dead Sea, pp 84–89
- Rabin M (1989) Efficient dispersal of information for security, load balancing, and fault tolerance. J ACM 36(2):335–348
- Rao TRN, Fujiwara E (1989) Error-control coding for computer systems. Prentice Hall, Upper Saddle River
- Rizzo L (1997) Effective erasure codes for reliable computer communication protocols. ACM Comput Commun Rev 27(2):24–36
- Schroeder B, Gibson GA (2007) Understanding disk failure rates: what does an MTTF of 1,000,000 hours mean to you? ACM Trans Storage 3(3):8, 31 pp
- Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. Nature 453(7191): 80–83
- Svrcek I (2009) Accelerated life cycle comparison of Millenniata archival DVD. Final report for Naval Air Warfare Center Weapons Division, 75 pp
- Yan W et al (2017) ROS: a rack-based optical storage system with inline accessibility for long-term data preservation. In: Proceedings of the 12th European conference on computer systems, Belgrade, pp 161–174

Definitions

According to Naumann (2013), Abedjan et al. (2015), data profiling is the set of activities and processes to determine the metadata about a given dataset. Metadata can range from simple statistics, such as the number of null values and distinct values in a column, its data type, or the most frequent patterns of its data values, to complex inter-value and intercolumn dependencies. Metadata that are more difficult to compute involve multiple columns, such as inclusion dependencies or functional dependencies. Collectively, a set of results of these tasks is called the *data profile* or *database profile*. In the *Encyclopedia of Database Systems*, Johnson considers data profiling as the activity of creating small but informative summaries of a database (Johnson 2009). In contrast to data mining, the focus of data profiling is typically the structure of a dataset. Thus data profiling generates knowledge about columns of a dataset, while data mining focusses on the actual records.

D

Data Management

- ▶ Big Data and Recommendation

Data Preparation

- ▶ Data Wrangling

Data Profiling

Ziawasch Abedjan
Technische Universität Berlin, Berlin, Germany

Synonyms

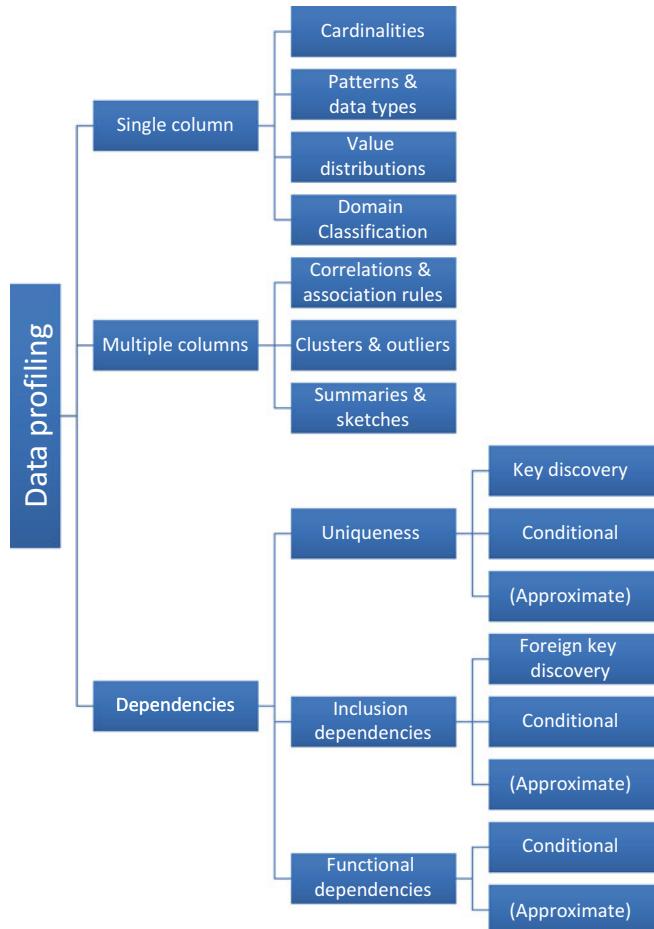
Metadata extraction; Metadata generation

Overview

Data profiling encompasses a vast array of methods to examine datasets and produce metadata. Like many data management tasks, data profiling faces three challenges: (i) managing the input, (ii) efficiently performing the computation, and (iii) managing the output. Apart from typical data formatting issues, the first challenge addresses the problem of specifying the expected outcome, i.e., determining which profiling tasks to execute on which parts of the data. In fact, many tools require a precise specification of what to inspect. Other approaches are more open and perform a wider range of tasks, discovering all metadata automatically.

Research and industry have tackled the second challenge in different ways, relying on the capabilities of the underlying DBMS and formulating profiling tasks as SQL queries, using indexing schemes, parallel processing, and reusing intermediate results. Additionally, several methods have been proposed that deliver only approximate

Data Profiling, Fig. 1 A classification of traditional data profiling tasks



results for various profiling tasks in order to handle Big Data. Figure 1 shows the classification of profiling tasks according to Abedjan et al. (2015), which includes single-column tasks, multicolumn tasks, and dependency detection. While dependency detection falls under multicolumn profiling, it is assigned a separate profiling class to emphasize this large and complex set of tasks.

The third challenge has not yet been the focus of broader research. Typically an aggregation of the huge amounts of meta-data in form of histograms or graphs is desirable (Kandel et al. 2012; Abedjan et al. 2014a; Papenbrock et al. 2015a).

Key Research Findings

There are different directions of research that address the three general categories of data pro-

filing. Research on single column profiling is mostly concerned with the reduction of complexity in the number of rows, trying to avoid full scans or superlinear operations on the entire column. Research on multicolumn profiling and dependency discovery is mostly concerned with the reduction of the exponential complexity with regard to the number of attribute and value combinations, trying to discover new practical pruning techniques.

Single-column profiling addresses the generation of counts, such as the number of values, the number of unique values, and the number of non-null values. These metadata are often part of the basic statistics gathered by the DBMS. In addition, the maximum and minimum values are discovered, and the data type is derived (usually restricted to string vs. numeric vs. date). More advanced techniques create histograms of

value distributions and identify typical patterns in the data values in the form of regular expressions (Raman and Hellerstein 2001). Among specializations of these techniques are methods to determine the frequency distribution of soundex code, n-grams, and others; the inverse frequency distribution, i.e., the distribution of the frequency distribution; or the entropy of the frequency distribution of the values in a column (Kang and Naughton 2003). A particularly interesting distribution is the first digit distribution for numeric values. Benford's law states that in naturally occurring numbers, the distribution of the first digit d of a number approximately follows $P(d) = \log_{10}(1 + \frac{1}{d})$ (Benford 1938).

Multicolumn profiling generalizes profiling tasks on single columns to multiple columns and also identifies inter-value dependencies and column similarities. There has been work on computing multidimensional histograms for query optimization (Poosala and Ioannidis 1997; Deshpande et al. 2001). Multicolumn profiling also plays an important role in data cleansing, e.g., in assessing and explaining data glitches, which can be exposed considering column combinations instead of single columns (Dasu et al. 2014). Another task is to identify correlations between values through frequent patterns or association rules (Hipp et al. 2000). Furthermore, clustering approaches that consume values of multiple columns as features allow for the discovery of coherent subsets of data records and outliers (Berti-Equille et al. 2011; Dasu and Loh 2012). Similarly, generating summaries and sketches of large datasets relates to profiling values across columns (Garofalakis et al. 2013; Ntarmos et al. 2009; Dasu et al. 2006).

Dependencies are metadata that describe relationships among columns. The difficulties of automatically detecting such dependencies in a given dataset are twofold: First, pairs of columns or larger column sets must be examined, and second, the existence of a dependency in the data at hand does not imply that this dependency is meaningful. While much research has been invested in addressing the first challenge, there is less work on semantically interpreting the profiling results. There has been a series of new algorithms that apply hybrid pruning techniques

that reduce the runtime of validating a dependency by using inverse indexing structures (Heise et al. 2013; Papenbrock et al. 2015b; Abedjan et al. 2014b). Additionally, new algorithms apply advanced search strategies to efficiently prune the exponential solution space, using random walk techniques or user-defined heuristics (Heise et al. 2013; Papenbrock and Naumann 2016). There is also a huge body of work on the so-called relaxed dependencies (Caruccio et al. 2016), which focus on dependencies that only hold (1) partially, i.e., for a subset of the data; (2) conditionally, i.e., for a subset of the data co-occurring with a condition; or (3) for similarity-based relationships, i.e., dependencies are not invalid if depending values are unequal as long as they are at least similar.

Examples of Application

Data profiling has many traditional use cases, including the data exploration, data cleansing, and data integration scenarios. Statistics about data are also useful in query optimization.

Data exploration. Users are often confronted with new unknown datasets. Examples include data files downloaded from the Web, old database dumps, or newly gained access to some DBMS. In many cases, such data have no known schema, no or old documentation, etc. Even if a formal schema is specified, it might be incomplete, for instance, specifying only the primary keys but no foreign keys. A natural first step is to identify the basic structure and high-level content of a dataset. Data profiling techniques can support such manual exploration. Simple, ad hoc SQL queries can reveal some insight, such as the number of distinct values, but more sophisticated methods are needed to efficiently and systematically discover metadata.

Database management. Typically, the generated metadata include various counts, such as the number of values, the number of unique values, and the number of non-null values. These metadata are often part of the basic statistics gathered by a DBMS. An optimizer uses them to estimate the selectivity of operators and perform

other optimization steps. Mannino et al. give a survey of statistics collection and its relationship to database optimization (Mannino et al. 1988). More advanced techniques use histograms of value distributions, functional dependencies, and unique column combinations to optimize range queries (Poosala et al. 1996) or for dynamic reoptimization (Kache et al. 2006).

Database reverse engineering. Given a “bare” database instance, the task of schema and database reverse engineering is to identify its relations and attributes, as well as domain semantics, such as foreign keys and cardinalities (Petit et al. 1994; Markowitz and Makowsky 1990). Hainaut et al. call these metadata “implicit constructs,” i.e., those that are not explicitly specified by DDL statements (Hainaut et al. 2009). However, possible sources for reverse engineering are DDL statements, data instances, data dictionaries, etc. The result of reverse engineering might be an entity-relationship model or a logical schema to assist experts in maintaining, integrating, and querying the database.

Data integration. Before different data sources can be integrated, one has to consider the dimensions of a dataset, its data types, and formats. A concrete use case for data profiling is that of *schema matching*, i.e., finding semantically correct correspondences between elements of two schemata (Euzenat and Shvaiko 2013). Many schema matching systems perform data profiling to create attribute features, such as data type, average value length, patterns, etc., to compare feature vectors and align those attributes with the best matching ones (Madhavan et al. 2001; Naumann et al. 2002).

Scientific data management and integration have created additional motivation for efficient and effective data profiling: When importing raw data, e.g., from scientific experiments or extracted from the Web, into a DBMS, it is often necessary and useful to profile the data and then devise an adequate schema. In many

cases, scientific data is produced by non-database experts and without the intention to enable integration. Thus, they often come with no adequate schematic information, such as data types, keys, or foreign keys. Additionally, many life-science databases have been created and updated through a long period of time creating disconnected and redundant tables that require exhaustive search and exploration before they can be put into purposeful use again.

Data quality/data cleansing. A typical use case of data profiling data is the *data cleansing* process. Commercial data profiling tools are usually bundled with corresponding data quality/data cleansing software. Profiling as a data quality assessment tool reveals data errors, such as inconsistent formatting within a column, missing values, or outliers. Profiling results can also be used to measure and monitor the general quality of a dataset, for instance, by determining the number of records that do not conform to previously established constraints (Pipino et al. 2002; Kandel et al. 2012). Generated constraints and dependencies also allow for rule-based data imputation.

Big Data analytics. Many Big Data and related data science scenarios call for data mining and machine learning techniques to explore and mine data. Again, data profiling is an important preparatory task to determine which data to mine, how to import it into the various tools, and how to interpret the results (Pyle 1999). In this context, leading researchers have noted “*If we just have a bunch of datasets in a repository, it is unlikely anyone will ever be able to find, let alone reuse, any of this data. With adequate metadata, there is some hope, but even so, challenges will remain[...]*”(Agrawal et al. 2012)

Future Directions for Research

Much of the data that currently is being focused on for application purposes is of nontra-

ditional type, i.e., non-relational, non-structured (textual), and heterogeneous. Many existing profiling methods cannot adequately handle that kind of data: Either they do not exist yet or they do not scale to the existing dimensions of data. Furthermore, fast data poses a new challenge to data profiling. With fast data we are considering data that changes over time through transactions or simply streaming data. For a more elaborate overview of upcoming challenges of data profiling, we refer to (Naumann 2013).

As mentioned in the introduction, the interpretation of profiling results remains an open challenge. To facilitate interpretation of metadata, effectively visualizing any profiling results is of utmost importance. So far, there are only prototypes that rudimentary tackle visualization of metadata (Kandel et al. 2012; Abedjan et al. 2014a; Papenbrock et al. 2015a).

References

- Abedjan Z, Grütze T, Jentzsch A, Naumann F (2014a) Profiling and mining RDF data with ProLOD++. In: Proceedings of the international conference on data engineering (ICDE), pp 1198–1201, Demo
- Abedjan Z, Schulze P, Naumann F (2014b) DFD: efficient functional dependency discovery. In: Proceedings of the international conference on information and knowledge management (CIKM), pp 949–958
- Abedjan Z, Golab L, Naumann F (2015) Profiling relational data: a survey. VLDB J 24(4):557–581
- Agrawal D, Bernstein P, Bertino E, Davidson S, Dayal U, Franklin M, Gehrke J, Haas L, Halevy A, Han J, Jagadish HV, Labrinidis A, Madden S, Papakonstantinou Y, Patel JM, Ramakrishnan R, Ross K, Shahabi C, Suciu D, Vaithyanathan S, Widom J (2012) Challenges and opportunities with Big Data. Technical report, Computing Community Consortium. <http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf>
- Benford F (1938) The law of anomalous numbers. Proc Am Philos Soc 78(4):551–572
- Berti-Equille L, Dasu T, Srivastava D (2011) Discovery of complex glitch patterns: a novel approach to quantitative data cleaning. In: Proceedings of the international conference on data engineering (ICDE), pp 733–744
- Caruccio L, Deufemia V, Polese G (2016) Relaxed functional dependencies – a survey of approaches. IEEE Trans Knowl Data Eng (TKDE) 28(1):147–165. <https://doi.org/10.1109/TKDE.2015.2472010>
- Dasu T, Loh JM (2012) Statistical distortion: consequences of data cleaning. Proc VLDB Endow (PVLDB) 5(11):1674–1683
- Dasu T, Johnson T, Marathe A (2006) Database exploration using database dynamics. IEEE Data Eng Bull 29(2):43–59
- Dasu T, Loh JM, Srivastava D (2014) Empirical glitch explanations. In: Proceedings of the international conference on knowledge discovery and data mining (SIGKDD), pp 572–581. ISBN: 978-1-4503-2956-9
- Deshpande A, Garofalakis M, Rastogi R (2001) Independence is good: dependency-based histogram synopses for high-dimensional data. In: Proceedings of the international conference on management of data (SIGMOD), pp 199–210. ISBN: 1-58113-332-4
- Euzenat J, Shvaiko P (2013) Ontology matching, 2nd edn. Springer, Berlin/Heidelberg/New York
- Garofalakis M, Keren D, Samoladas V (2013) Sketch-based geometric monitoring of distributed stream queries. Proc VLDB Endow (PVLDB) 6(10):937–948
- Hainaut J-L, Henrard J, Englebert V, Roland D, Hick J-M (2009) Database reverse engineering. In: Encyclopedia of database systems. Springer, Heidelberg, pp 723–728
- Heise A, Quiané-Ruiz J-A, Abedjan Z, Jentzsch A, Naumann F (2013) Scalable discovery of unique column combinations. Proc VLDB Endow (PVLDB) 7(4): 301–312
- Hipp J, Güntzer U, Nakhaeizadeh G (2000) Algorithms for association rule mining – a general survey and comparison. SIGKDD Explor 2(1):58–64. ISSN: 1931-0145
- Johnson T (2009) Data profiling. In: Encyclopedia of database systems. Springer, Heidelberg, pp 604–608
- Kache H, Han W-S, Markl V, Raman V, Ewen S (2006) POP/FED: progressive query optimization for federated queries in DB2. In: Proceedings of the international conference on very large databases (VLDB), pp 1175–1178
- Kandel S, Parikh R, Paepcke A, Hellerstein J, Heer J (2012) Profiler: integrated statistical analysis and visualization for data quality assessment. In: Proceedings of advanced visual interfaces (AVI), pp 547–554
- Kang J, Naughton JF (2003) On schema matching with opaque column names and data values. In: Proceedings of the international conference on management of data (SIGMOD), pp 205–216
- Madhavan J, Bernstein PA, Rahm E (2001) Generic schema matching with Cupid. In: Proceedings of the international conference on very large databases (VLDB), pp 49–58
- Mannino MV, Chu P, Sager T (1988) Statistical profile estimation in database systems. ACM Comput Surv 20(3):191–221
- Markowitz VM, Makowsky JA (1990) Identifying extended entity-relationship object structures in relational schemas. IEEE Trans Softw Eng 16(8):777–790
- Naumann F (2013) Data profiling revisited. SIGMOD Rec 42(4):40–49

- Naumann F, Ho C-T, Tian X, Haas L, Megiddo N (2002) Attribute classification using feature analysis. In: Proceedings of the international conference on data engineering (ICDE), p 271
- Ntarmos N, Triantafillou P, Weikum G (2009) Distributed hash sketches: scalable, efficient, and accurate cardinality estimation for distributed multisets. ACM Trans Comput Syst (TOCS) 27(1):1–53. ISSN:0734-2071
- Papenbrock T, Naumann F (2016) A hybrid approach to functional dependency discovery. In: Proceedings of the international conference on management of data (SIGMOD), pp 821–833. <https://doi.org/10.1145/2882903.2915203>, <http://doi.acm.org/10.1145/2882903.2915203>
- Papenbrock T, Bergmann T, Finke M, Zwiener J, Naumann F (2015a) Data profiling with metanome. Proc VLDB Endow (PVLDB) 8:1860–1863
- Papenbrock T, Kruse S, Quiané-Ruiz J-A, Naumann F (2015b) Divide & conquer-based inclusion dependency discovery. Proc VLDB Endow (PVLDB) 8(7):774–785
- Petit J-M, Kouloumdjian J, Boulicaut J-F, Toumani F (1994) Using queries to improve database reverse engineering. In: Proceedings of the international conference on conceptual modeling (ER), pp 369–386
- Pipino L, Lee Y, Wang R (2002) Data quality assessment. Commun ACM 4:211–218
- Poosala V, Ioannidis YE (1997) Selectivity estimation without the attribute value independence assumption. In: Proceedings of the international conference on very large databases (VLDB), pp 486–495
- Poosala V, Haas PJ, Ioannidis YE, Shekita EJ (1996) Improved histograms for selectivity estimation of range predicates. In: Proceedings of the international conference on management of data (SIGMOD), pp 294–305
- Pyle D (1999) Data preparation for data mining. Morgan Kaufmann, San Francisco
- Raman V, Hellerstein JM (2001) Potters wheel: an interactive data cleaning system. In: Proceedings of the international conference on very large databases (VLDB), pp 381–390

Data Provenance for Big Data Security and Accountability

Thye Way Phua and Ryan K. L. Ko
Cyber Security Lab – Department of Computer Science, University of Waikato, Hamilton, New Zealand

Synonyms

[Lineage](#); [Pedigree](#)

Definitions

Provenance is the derivative history of data (Ko et al. 2015; Ko and Will 2014). While provenance does not directly contribute to upholding and enforcing the information security requirements (confidentiality, integrity, and availability) in the context of Big Data security, provenance and its sources (e.g., metadata, lineage, data activities (create, read, update, and delete)) strongly provide verification and historical evidence to support the analysis or forecasting needs for the purpose of data security. One example is to analyze provenance to understand and prevent outages better (Ko et al. 2012), so as to achieve better availability. Provenance also contributes strongly to data forensics, especially in the study of data activity patterns triggered by software or human processes (Ko et al. 2015) (e.g., ransomware). The lineage and metadata describing provenance also provide substantial evidence for transparency and data accountability (Ko 2014). With provenance, one would essentially be able to find out or verify “what happened behind the scenes” for Big Data activities, leading to a strong case for security attribution.

Overview

In the process of Big Data leveraging the high speed (velocity), large amount (volume), and various types (variety) of data item to provide better analytics (value), there is an inherent heavily reliance on data quality (veracity). Consequently, Big Data faces challenges of data integrity, data quality, and privacy. In perhaps a cyclical way, these challenges are in turn amplified by the 4V characteristics: velocity, volume, variety, and veracity.

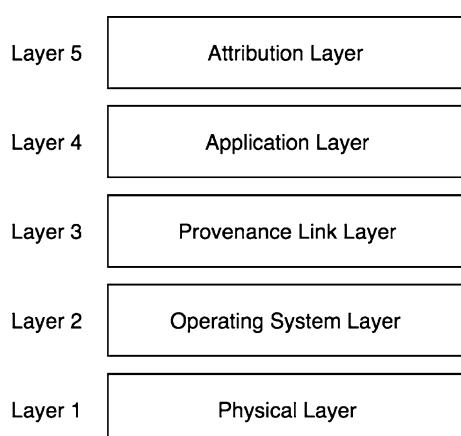
A typical Big Data workflow consists of five phases: data acquisition, extraction, integration, analysis, and interpretation. There are security risks associated with each phase, and these risks greatly affect the soundness and effectiveness of Big Data analysis. Data provenance, the deriva-

tion history of data (Ko et al. 2015; Ko and Will 2014), can improve the quality of the results by providing supporting metadata to enhance the Big Data workflow and to also serve as a tool for security, attribution, and forensics. While data provenance does not provide the security requirements of confidentiality, integrity, and availability, it serves as a security tool to provide detection, prevention, and mitigation needs – increasing the accountability of Big Data.

Provenance Abstraction Layers

Using the Full Provenance Stack (Ko and Phua 2017) shown in Fig. 1 as a guideline, a complete and meaningful Big Data provenance can be achieved. The Full Provenance Stack is an abstract model that classifies provenance into five separate layers. The higher layers encapsulate the layer below to refine the understanding of provenance from layers below them. Layer 1 is the physical layer. In this layer, provenance information found at hardware and BIOS/firmware level can be collected and analyzed. This is especially relevant in verifying the hardware integrity of the data-generating source. Data originating from a source that fails to establish its integrity can be discarded. In layer 2, data provenance in layer 1 is encapsulated to provide atomic human-readable provenance data,

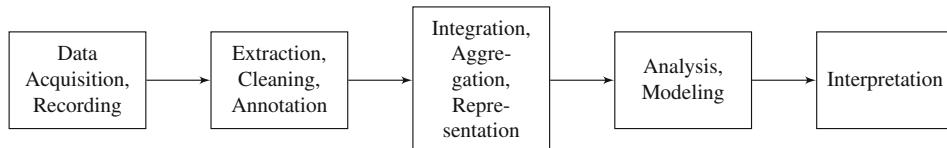
at the operating systems running on the layer 1 hardware. Activities such as data creation, reading, updating, and deleting are now captured in atomic actions such as operating system calls. Layer 3 of the provenance stack makes sense of the data flow tracking of ingress and egress of Big Data information flow between and within nodes. Nodes may be virtual machines, routers, mobile devices, or physical servers. This layer ensures there is an accurate encapsulation and representation of data provenance in between and across nodes. A compromised node can be identified by verifying the layer 3 provenance. Layer 4 provenance is what most Big Data systems record, the application provenance. This is provenance specific to a particular application or a specific phase in the Big Data workflow. Examples can be Web browsing information or database transactions. This layer of provenance provides phase-specific, semantic-rich provenance data. Layer 5, the final layer of the provenance stack, is the attribution layer; this is the encapsulation of all layers below, to provide a complete understanding of the who, what, when, where, how, and why explanation of the analysis results. At layer 5, we can attribute data creators and provide accountability and transparency for Big Data.



Data Provenance for Big Data Security and Accountability, Fig. 1 Full provenance stack (Ko and Phua 2017)

Provenance Phases in Big Data Workflows

From understanding the layers, we now move to understanding the flow and phases of provenance in relation to the Big Data workflow shown in Fig. 2. The data acquisition phase revolves around the gathering of data from various data-generating sources. This could be a heart rate monitor, emails, social network, or any Internet of things (IoT) devices. The key challenge in this phase is data quality. As the volume of data acquired may be enormous, in many cases, most of the data is discarded, either because it appears to be erroneous or of no relevance. However, it is difficult to justify which and why data should be discarded. For example, a data set may seem irrelevant at a moment but may in reality play a crucial role in the analysis process at a later



Data Provenance for Big Data Security and Accountability, Fig. 2 Big data workflow (Agrawal et al. 2012)

stage. Data provenance can record and provide the metadata required to strengthen such justifications. For example, the provenance metadata can provide information relating to how the data set was recorded, in what unit was it measured in, and further details of the data-generating source. In addition to that, data provenance can also help to eliminate inputs from untrusted data-generating sources. For example, after identifying a source that has been providing inconsistent, missing, or erroneous data, a filter may be put in place to analyze the provenance metadata and discard any data originating from this untrusted source.

Next, we discuss the extraction phase. Because data sets come in myriad forms – from sensor data to images or plain text – they may sometimes be annotated or prepared prior to analysis. The extraction phase attempts to extract information into structured form to ease analysis. An example of data extraction method is natural-language processing (NLP). This process concerns the data quality and data integrity, that is, how complete and accurate the extracted data is. Data provenance has two responsibilities here: it provides the necessary metadata to provide context and semantics to allow for better and more accurate extraction process; it also records a new provenance, to support the other phases in the pipeline. Data provenance also empowers better data integrity – aligned to typical information security requirements.

The integration phase focuses on incorporating data from other internal or external sources to expand the possibilities of Big Data analysis, for example, relating a customer's shopping habits to their online social network activities. The relationship between these data sets is not explicit and can be challenging; relevant data may be excluded resulting in missed opportunities; or

irrelevant data may be included resulting in a less accurate analytic result(s). Data provenance may reveal some non-apparent relationships between two data sets to make the integration process more coherent. Apart from relevance, the quality of the integrated data needs to be verified to prevent contamination of data. The system can gauge the trustworthiness and quality of integrated data set through their provenance.

After the analysis phase, the results are being interpreted in the interpretation phase. The interpretation phase is not a straightforward process. One must not only possess the skill but must also take into account system errors and the assumptions made. Results produced need to be verified and justified. Data provenance aids in interpreting the results by providing information on (1) how the result was derived, (2) which data contributes to the result, and (3) how a particular data set contributes to the result. Data provenance information should reveal any possibilities or at least make identifying errors and false assumptions clearer. Therefore, one can repeat the analysis process to obtain a more accurate result.

In addition, the distributed environment and actors are also of concern in Big Data security. Big Data analysis is usually carried out in a cloud environment due to their capabilities and flexibilities. However, using a third-party computing and storage resource like cloud computing poses a risk to the privacy of data, data integrity, and data quality. The security of the cloud provider needs to be taken into account, internally and externally. An external risk is that an attacker can either steal or perform malicious attack to degrade the quality of the results. An internal risk can be a malicious insider at the cloud provider. In this case, provenance information could enable the transparency required to prevent security risks in the future.

Key Provenance Research for Big Data Security

One of the earliest research applications for data provenance in distributed, large-scale Big Data was in the area of scientific workflows. Data provenance information is used by the scientific community to facilitate the reuse and sharing of experimental data. The use of data provenance has then been extended to several areas such as operating systems, database systems, and cloud computing.

Provenance-aware storage system (PASS) (Muniswamy-Reddy et al. 2006) is a storage system that automatically collects and maintains provenance. Muniswamy-Reddy et al. proposed that provenance should be treated as metadata collected and maintained by storage system. This allows storage system to generate system-level provenance automatically while easing the need for additional provenance management task. This system captures provenance data by intercepting system calls. A stackable file system was used along with a database engine to store provenance metadata.

File-Centric Logger (Flogger) (Ko et al. 2011) by Hewlett-Packard Labs was the first data-centric provenance logger for cloud data provenance tracking. This system uses the Linux Loadable Kernel Module and the Windows Device Driver to capture cloud data provenance by intercepting file and network operations. Provenance was captured at all virtual machines and physical machines. The provenance from virtual machines was securely transmitted to their physical machines via a communication channel and subsequently written to log files and then stored in a data store. Provenance from virtual machines can be correlated with the related provenance captured in the physical machines hosting the virtual machines. Ko et al. have identified that this system, like most system, runs under the assumption of kernel integrity and that provenance logs need to be tamper-proof and immutable.

Provenance logger (Progger) (Ko and Will 2014) is a tamper-evident, kernel-space cloud data provenance system. The system was de-

signed to address four critical data security and integrity problems: tamper-evidence of provenance logs, precise timestamp synchronization across machines, growth of provenance logs, and the efficient logging of root usage of the system. This work later inspired HPLogger (Fu et al. 2017), which is a provenance monitoring tool built upon Progger, tailored specific for Hadoop. HPLogger reduces the number of system calls recorded, minimized the log files by reducing the details captured, and limits the capturing scope within the Hadoop working directory. While this improves the performance of the provenance logging system, the granularity and level of details are also significantly reduced.

D

Examples Applications

Identification and exclusion of untrusted source. In Big Data, it is common for data from different sources and origins to be integrated (data fusion). However, if one or multiple combined source(s) of data is of low quality, it is hard to filter them out after it has been combined and aggregated. Data provenance is able to clearly identify each data unit and allows for removal of data from untrusted source.

Combating data poisoning attacks. Data poisoning attack is the deliberate introduction of false data to affect the Big Data analytic results. Such data can be filtered out and discarded with the help of data provenance. The origin of the data can be verified from the data provenance, and even if the origin has been spoofed, by comparing other provenance, metadata should reveal when and how was the false data introduced. For example, the false data may have missing data provenance information on extraction phase of the Big Data workflow; this can be an indicator that this is a false data, and this data is introduced after the extraction phase.

Audit trails identifying source of problems. In distributed environment, a processing node may be compromised or contains bugs, therefore

contaminating the analysis of the results. An error introduced in any phase of the Big Data workflow will render the entire process a failure. Data provenance keeps a complete audit trail, allowing for clear explanation of the source and cause of the problem, that is in which phase the error occurred, which nodes and data is affected and how are they affected.

Future Directions for Research

First-person provenance. Existing provenance logging systems can be regarded as third-person provenance, i.e., the host which is processing the data monitors the actions performed on the data and records it in a log file or database, separated from the data itself. Since Big Data analysis is performed in a distributed environment, recording the creation and modification isolated within an environment cannot provide a full picture unless the provenance data of all systems involved are aggregated. In other words, the provenance generated is only useful if it is recorded and passed down the Big Data workflow pipeline. This allows each phase to utilize the provenance recorded. Therefore, a first-person provenance approach is needed, such that data provenance is embedded along with the data itself (e.g., within the file which describes the data) and exists throughout the entire Big Data workflow.

Secure provenance. As expressed in the section above, data provenance can be used as a security tool. However, this is under the assumption that the provenance tracking and reporting system is accurate and trusted. Similar to any system, provenance system can be manipulated and compromised to record and report incorrect information or even leak sensitive information. Therefore, in order to rely on data provenance as a security tool, the provenance system must record accurate provenance, tamper-proof or tamper-evident, and enforce dynamic, fine-grained access control. Current provenance systems have yet to achieve or have only partially achieved these requirements.

Provenance storage overheads. One of the main challenges to collecting provenance is its overheads, particularly storage overheads. Provenance can grow up to 22 times the size of the original data (Xie et al. 2013). There have been proposals for provenance compression, but compression and decompression require computing effort and may shift the problem of storage overhead to performance overheads. This is especially crucial in Big Data as both storage and performance are critical.

Cross-References

- ▶ [Data Quality and Data Cleansing of Semantic Data](#)
- ▶ [Secure Big Data Computing in Cloud: an Overview](#)

References

- Agrawal D, Bernstein P, Bertino E, Davidson S, Dayal U, Franklin M, Gehrke J, Haas L, Halevy A, Han J, Jagadish HV, Labrinidis A, Madden S, Papakonstantinou Y, Patel JM, Ramakrishnan R, Ross K, Shahabi C, Suciu D, Vaithyanathan S, Widom J (2012) Challenges and opportunities with big data: a white paper prepared for the computing community consortium committee of the Computing Research Association. Technical report. <https://cra.org/ccc/wp-content/uploads/sites/2/2015/05/bigdatawhitepaper.pdf>
- Fu X, Gao Y, Luo B, Du X, Guizani M (2017) Security threats to hadoop: data leakage attacks and investigation. *IEEE Netw* 31(2):67–71
- Ko RKL (2014) Data accountability in cloud systems. In: Nepal S, Pathan M (eds) *Security, privacy and trust in cloud systems*. Springer, Berlin. pp 211–238
- Ko RKL, Phua TW (2017) The full provenance stack: five layers for complete and meaningful provenance. In: *Proceedings of the security, privacy and anonymity in computation, communication and storage: SpaCCS 2017 international workshops, UbiSafe, ISSR, Trust-Data, TSP, SPIoT, NOPE, DependSys, SCS, WC-SSC, MSCF and SPBD, 12–15 Dec 2017*. Springer, Guangzhou
- Ko RKL, Will MA (2014) Progger: an efficient, tamper-evident kernel-space logger for cloud data provenance tracking. In: *Proceedings of the IEEE international conference on cloud computing, CLOUD ’14*. IEEE Computer Society, Washington, DC, pp 881–889. <https://doi.org/10.1109/CLOUD.2014.121>

- Ko RKL, Jagadpramana P, Lee BS (2011) Flogger: a file-centric logger for monitoring file access and transfers within cloud computing environments. In: Proceedings of the IEEE 10th international conference on trust, security and privacy in computing and communications, TRUSTCOM '11. IEEE Computer Society, Washington, DC, pp 765–771. <https://doi.org/10.1109/TrustCom.2011.100>
- Ko RKL, Lee SSG, Rajan V (2012) Understanding cloud failures. IEEE Spectr 49(12):84–84. <https://doi.org/10.1109/MSPEC.2012.6361788>
- Ko RKL, Russello G, Nelson R, Pang S, Cheang A, Dobbie G, Sarrafzadeh A, Chaisiri S, Asghar MR, Holmes G (2015) Stratus: towards returning data control to cloud users. In: International conference on algorithms and architectures for parallel processing. Springer, pp 57–70
- Muniswamy-Reddy KK, Holland DA, Braun U, Seltzer MI (2006) Provenance-aware storage systems. In: USENIX annual technical conference, general track. pp 43–56
- Xie Y, Muniswamy-Reddy KK, Feng D, Li Y, Long DDE (2013) Evaluation of a hybrid approach for efficient provenance storage. Trans Storage 9(4):14:1–14:29. <https://doi.org/10.1145/2501986>

Data Quality

- ▶ [Data Quality and Data Cleansing of Semantic Data](#)

Data Quality and Data Cleansing of Semantic Data

Amrapali Zaveri¹ and Anisa Rula²

¹Institute of Data Science, Maastricht University, Maastricht, The Netherlands

²Department of Computer Science, Systems and Communication (DISCo), University of Milano-Bicocca, Milan, Italy

Synonyms

[Data cleaning](#); [Data curation](#); [Data quality](#); [Data validation](#); [Linked data](#); [Quality assessment](#)

Definitions

Data quality is commonly conceived as a multidimensional construct defined as “fitness for use.” Data quality may depend on various factors (dimensions or characteristics) such as completeness, consistency, availability, etc. Data quality assessment involves the measurement of quality dimensions or criteria that are relevant to the use case. A metric or measure is a procedure for measuring a data quality dimension with the help of a tool. These metrics are heuristics that are designed to fit a specific assessment situation.

Overview

In this chapter, we first introduce the concepts of Linked Data quality and its dimensions and metrics. Then we provide definitions for 18 quality dimensions along with a total of 69 metrics to measure the dimensions.

Thereafter, we provide an overview of tools currently available for Linked Data quality assessment followed by an introduction to the W3C Data Quality Vocabulary. Finally, we discuss some of the open research challenges along with few solutions already available.

Semantic Web Data Quality

The increasing diffusion of the Semantic Web as a standard way to share knowledge on the Web allows users and public and private organizations to fully exploit structured data from very large datasets, which were not available in the past. Over the last few years, Linked Data (LD) developed into a large number of datasets with an open access from several domains leading to the Linked Open Data cloud (Heath and Bizer 2011). This vast rich integrate data space can fuel many data-driven discoveries; however, this process is quite challenging due to the (poor) quality of Linked Data. However, assessing Linked Data quality is particularly a challenge as the underlying data stems from a set of multiple, autonomous, and evolving data sources. It is

extremely important to assess the quality of the datasets used in Linked Data applications, before using them, allowing users or applications to understand whether data is appropriate for their task at hand.

Data Quality Dimensions and Metrics

Data quality assessment involves the measurement of quality *dimensions* or *criteria* that are relevant to the consumer. The dimensions can be considered as the characteristics of a dataset. A data quality assessment *metric*, *measure*, or *indicator* is a procedure for measuring a data quality dimension (Bizer and Cyganiak 2009). These metrics are heuristics that are designed to fit a specific assessment situation (Leo Pipino and Rybold 2005). An assessment score is computed from these indicators using a scoring function. In the next section, definitions of 18 data quality dimensions are provided along with their respective 69 metrics that are studied in a survey of the quality of Linked Data (performed in Zaveri et al. 2016). These dimensions are classified into the (i) accessibility, (ii) intrinsic, (iii) contextual, and (iv) representational groups.

Accessibility Dimensions

The dimensions belonging to this category involve aspects related to the access, authenticity, and retrieval of data to obtain either the entire or some portion of the data (or from another linked dataset) for a particular use case. There are five dimensions that are part of this group, which are *availability*, *licensing*, *interlinking*, *security*, and *performance*.

Availability of a dataset is the extent to which data (or some portion of it) is present, obtainable, and ready for use.

Metrics. Availability of a dataset can be measured in terms of accessibility of the server, SPARQL endpoints, or RDF dumps and also by the dereferencability of the URIs.

Licensing is defined as the granting of permission for a consumer to reuse a dataset under defined conditions.

Metrics. Licensing can be checked by the indication of machine- and human-readable information associated with the dataset clearly indicating the permissions of data reuse.

Interlinking refers to the degree to which entities that represent the same concept are linked to each other, be it within or between two or more data sources.

Metrics. Interlinking can be measured by using network measures that calculate the interlinking degree, cluster coefficient, sameAs chains, centrality, and description richness through sameAs links.

Security is the extent to which data is protected against alteration and misuse.

Metrics. Security can be measured based on whether the data has a proprietor or requires web security techniques (e.g., SSL or SSH) for users to access, acquire, or reuse the data. The importance of security depends on whether the data needs to be protected and whether there is a cost of data becoming unintentionally available. For open data, the protection aspect of security can be often neglected, but the non-repudiation of the data is still an important issue. Digital signatures based on private-public key infrastructures can be employed to guarantee the authenticity of the data.

Performance refers to the efficiency of a system that binds to a large dataset, that is, the more performant a data source is, the more efficiently a system can process data.

Metrics. Performance is measured based on the scalability of the data source, that is, a query should be answered in a reasonable amount of time. Also, detection of the usage of prolix RDF features or usage of slash URIs can help deter-

mine the performance of a dataset. Additional metrics are low latency and high throughput of the services provided for the dataset.

Intrinsic Dimensions

Intrinsic dimensions are those that are independent of the user's context. There are five dimensions that are part of this group, which are *syntactic validity*, *semantic accuracy*, *consistency*, *conciseness*, and *completeness*. These dimensions focus on whether information correctly (syntactically and semantically), compactly, and completely represents the real world and whether information is logically consistent in itself.

Syntactic validity is defined as the degree to which an RDF document conforms to the specification of the serialization format.

Metrics. A syntax validator can be employed to assess the validity of a document, i.e., its syntactic correctness. The syntactic accuracy of entities can be measured by detecting the erroneous or inaccurate annotations, classifications, or representations. An RDF validator can be used to parse the RDF document and ensure that it is syntactically valid, that is, to check whether the document is in accordance with the RDF specification.

Semantic accuracy is defined as the degree to which data values correctly represent the real-world facts.

Metrics. Accuracy can be measured by checking the correctness of the data in a data source, that is, the detection of outliers or identification of semantically incorrect values through the violation of functional dependency rules.

Consistency means that a knowledge base is free of (logical/formal) contradictions with respect to particular knowledge representation and inference mechanisms.

Metrics. On the Linked Data Web, semantic knowledge representation techniques are employed, which come with certain inference and reasoning strategies for revealing implicit knowledge, which then might render a contradiction. Consistency is relative to a particular logic (set of inference rules) for identifying contradictions. A consequence of the definition of consistency is that a dataset can be consistent wrt. the RDF inference rules but inconsistent when taking the OWL2-QL reasoning profile into account. For assessing consistency, one can employ an inference engine or a reasoner, which supports the respective expressivity of the underlying knowledge representation formalism. Additionally, one can detect functional dependency violations such as domain/range violations. In practice, RDF Schema inference and reasoning with regard to the different OWL profiles can be used to measure consistency in a dataset.

Conciseness refers to the minimization of redundancy of entities at the schema and the data level. Conciseness is classified into (i) intensional conciseness (schema level) which refers to the case when the data does not contain redundant schema elements (properties and classes) and (ii) extensional conciseness (data level) which refers to the case when the data does not contain redundant objects (instances).

Metrics. As conciseness is classified in two categories, it can be measured as the ratio between the number of unique attributes (properties) or unique objects (instances) and the overall number of attributes or objects, respectively, present in a dataset.

Completeness refers to the degree to which all required information is present in a particular dataset. In terms of LD, completeness comprises of the following aspects:

Schema completeness the degree to which the classes and properties of an ontology are represented and thus can be called ontology completeness,

Property completeness measure of the missing values for a specific property,

Population completeness is the percentage of all real-world objects of a particular type that are represented in the datasets and

Interlinking completeness, which has to be considered especially in LD, refers to the degree to which instances in the dataset are interlinked.

Metrics. Completeness can be measured by detecting the number of classes, properties, values, and interlinks that are present in the dataset by comparing it to the original dataset (or gold standard dataset). It should be noted that in this case, users should assume a closed-world assumption where a gold standard dataset is available and can be used to compare against. Completeness can be measured by detecting the number of classes, properties, values, and interlinks that are present in the dataset by comparing it to the original dataset (or gold standard dataset). It should be noted that in this case, users should assume a closed-world assumption where a gold standard dataset is available and can be used to compare against.

Contextual Dimensions

Contextual dimensions are those that highly depend on the context of the task at hand. There are four dimensions that are part of this group, namely, *relevancy*, *trustworthiness*, *understandability*, and *timeliness*.

Relevancy refers to the provision of information which is in accordance with the task at hand and important to the users' query.

Metrics. Relevancy is highly context dependent and can be measured by using meta-information attributes for assessing whether the content is relevant for a particular task. Additionally, retrieval of relevant documents can be performed using a combination of hyperlink analysis and information retrieval methods.

Trustworthiness is defined as the degree to which the information is accepted to be correct, true, real, and credible.

Metrics. The provenance information can in turn be used to assess the trustworthiness, reliability, and credibility of a data source, an entity, publishers, or even individual RDF statements. There exists an interdependency between the data provider and the data itself. On the one hand, data is likely to be accepted as true if it is provided by a trustworthy provider. On the other hand, the data provider is trustworthy if it provides true data. Thus, both can be checked to measure the trustworthiness.

Understandability refers to the ease with which data can be comprehended without ambiguity and be used by a human information consumer.

Metrics. Understandability can be measured by detecting whether human-readable labels for classes, properties, and entities are provided. Provision of the metadata of a dataset can also contribute toward assessing its understandability. The dataset should also clearly provide exemplary URIs and SPARQL queries along with the vocabularies used so that users can understand how it can be used.

Timeliness measures how up-to-date data is relative to a specific task.

Metrics. Timeliness is measured by combining the two dimensions: currency and volatility. Additionally timeliness states the recency and frequency of data validation and does not include outdated data.

Representational Dimensions

Representational dimensions capture aspects related to the design of the data such as the *representational conciseness*, *interoperability*, *interpretability*, as well as *versatility*.

Rep. conciseness refers to the representation of the data, which is compact and well formatted on the one hand and clear and complete on the other hand.

Metrics. Representational conciseness is measured by qualitatively verifying whether the RDF model that is used to represent the data is concise enough in order to be self-descriptive and unambiguous.

Interoperability is the degree to which the format and structure of the information conform to previously returned information as well as data from other sources.

Interpretability refers to technical aspects of the data, that is, whether information is represented using an appropriate notation and whether the machine is able to process the data.

Metrics. Interpretability can be measured by the use of globally unique identifiers for objects and terms or by the use of appropriate language, symbols, units, and clear definitions.

Versatility refers to the availability of the data in different representations and in an internationalized way.

Metrics. Versatility can be measured by the availability of the dataset in different serialization formats, different languages, as well as different access methods.

Linked Data Quality Assessment Tools

There exist several approaches toward developing frameworks for assessing the data quality of LD. These frameworks can be broadly classified into: (i) automated (e.g., Guéret et al. 2011), (ii) semi-automated (e.g., Kontokostas et al. 2014), or (iii) manual (e.g., Bizer and Cyganiak 2009) methodologies. These approaches are useful at the process level wherein they introduce a methodology to assess the quality of a dataset. Additionally, a new generation of assessment tools is emerging which enable the integration of automatic tasks with human tasks in a crowdsourcing approach for LD quality assessment. Crowdsourcing is highly appropriate for any assignment involving large to huge numbers of small tasks that require human judgment. In terms of LD, crowdsourcing quality assessment may involve, for example, verifying the completeness or correctness of a fact wrt. the original dataset. Such a task does not require underlying knowledge about the structure of the data and can be done fairly quickly, without bias, and cost-effectively. In one study (Acosta et al. 2013), a comparison between assessments done by LD experts (those conversant with RDF) and Amazon Mechanical Turk workers on the DBpedia dataset was undertaken. The study showed promising results, in terms of time as well as cost-efficiency, for using a combination of crowdsourcing as well as manual (by experts) methodologies for quality assessment.

Representing Quality Metadata as Linked Data: The W3C Data Quality Vocabulary

The W3C working group on the *Data on the Web Best Practice* asserts the importance of providing quality information of a dataset (linked or not) to consumers. In an effort toward standardizing a vocabulary to represent quality metadata for data, the Data on the Web Best Practices Working Group developed the Data Quality Vocabulary

(DQV) (Albertoni et al. 2015). The idea of this vocabulary is to describe various quality aspects of the attached data in a structured, interoperable, and human - readable format. The Data Quality Vocabulary extends the Data Catalog Vocabulary (DCAT) (Maali et al. 2014) with a number of concepts (classes and properties) suitable to express quality metadata.

Open Research Challenges

Large-Scale Quality Assessment

Current tools that perform quality assessment of Web data suffer from severe deficiencies in terms of performance, once the dataset size grows beyond the capabilities of a single machine. Recently, an approach for quality assessment evaluation of large RDF datasets has been implemented using the SANSA framework (Lehmann et al. 2017), which scales over clusters of machines. The approach uses distributed in-memory for computing different quality metrics for RDF datasets using Apache Spark. However, such systems still lack inference-aware knowledge graph embeddings, distributed reasoning, and intuitive data partitioning strategies.

Data Profiling for the Web of (Big) Data

Data profiling is the set of activities and processes to produce metadata about the content of a dataset. Data profiling provides a first understanding of the data through statistics and other useful metadata. The metadata, in turn, enables reuse of existing data in useful applications. The ABSTAT tool (Spahiu et al. 2016) uses a schema-based summarization approach to profile LOD, which provides summaries that reveal quality issues that can be further analyzed. However, current datasets lack high-quality metadata, which is extremely important to enable data reuse. To tackle this problem, the recently published FAIR principles (Wilkinson et al. 2016) have been established to guide data producers to ensure that their data is maximally findable, accessible, interoperable, and reusable. These principles apply to all digital research objects (e.g., repositories, datasets, APIs, databases, publications). There is

a need to analyze the currently available digital resources with these FAIR principles.

Quality Improvement of Published Data

In addition to the assessment of quality, there need to be methods and tools for the consequent improvement of the datasets. One important step to improve the quality of data is identifying the root cause of the problem and then designing corresponding data improvement solutions. These solutions would select the most effective and efficient strategies and related set of techniques and tools to improve quality. Currently, only the RDFUnit tool Kontokostas et al. (2014) reports the exact triples which causes the quality issue. However, there need to be systems in place which close the loop, per say, from data quality assessment to improvement.

Cross-References

- ▶ [Data Cleaning](#)
- ▶ [Data Profiling](#)
- ▶ [Linked Data Management](#)

References

- Acosta M, Zaveri A, Simperl E, Kontokostas D, Auer S, Lehmann J (2013) Crowdsourcing linked data quality assessment. In: Proceedings of the 12th international semantic web conference (ISWC). Lecture notes in computer science, vol 8219. Springer, Berlin/Heidelberg, pp 260–276
- Albertoni R, Isaac A, Guéret C, Debattista J, Lee D, Mihindukulasooriya N, Zaveri A (2015) Data quality vocabulary (DQV). W3c interest group note, World Wide Web consortium (W3C)
- Bizer C, Cyganiak R (2009) Quality-driven information filtering using the WIQA policy framework. J Web Semant 7(1):1–10
- Guéret C, Groth P, Stadler C, Lehmann J (2011) Linked data quality assessment through network analysis. In: The semantic web – ISWC 2011. Lecture notes in computer science, vol 7032. Springer, Berlin/Heidelberg
- Heath T, Bizer C (2011) Linked data: evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology, 1st edn. Morgan & Claypool, Milton Keynes
- Kontokostas D, Westphal P, Auer S, Hellmann S, Lehmann J, Cornelissen R (2014) Databugger: a test-driven framework for debugging the web of data. In:

- Proceedings of the companion publication of the 23rd international conference on world wide web companion, WWW companion '14, pp 115–118. <https://doi.org/10.1145/2567948.2577017>
- Lehmann J, Sejdiu G, Bümann L, Westphal P, Stadler C, Ermilov I, Bin S, Chakraborty N, Saleem M, Ngonga Ngomo AC, Jabeen H (2017) Distributed semantic analytics using the SANSA stack. In: Proceedings of the 16th international semantic web conference, part II, the semantic web – ISWC 2017, 21–25 Oct 2017. Springer International Publishing, Vienna
- Pipino L, Kopsco D, Wang R, Rybold W (2005) Developing measurement scales for data-quality dimensions, vol 1. M.E. Sharpe, New York
- Maali F, Erickson J, Archer P (2014) Data catalog vocabulary. W3C recommendation, world wide web consortium (W3C)
- Spahiu B, Porrini R, Palmonari M, Rula A, Maurino A (2016) ABSTAT: ontology-driven linked data summaries with pattern minimalization. In: The semantic web: ESWC 2016 satellite events, Heraklion, 29 May–2 June 2016. Springer International Publishing, pp 381–395
- Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten JW, da Silva Santos LB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJ, Goble C, Jeffrey S, Grethe PG, Heringa J, 't Hoen PA, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone SA, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, van der Lei J, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B (2016) The FAIR guiding principles for scientific data management and stewardship. *Scientific data* 3. <https://doi.org/10.1038/sdata.2016.18>
- Zaveri A, Rula A, Maurino A, Pietrobon R, Lehmann J, Auer S (2016) Quality assessment for linked data: a survey. *Semant Web J* 7:63–93

Data Replication and Encoding

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

[Big Data Availability; Incorruptible big data](#)

Definition of Entry

Conventional or well-established redundancy methods for preventing data loss, unavailability, or corruption can be used to protect big data, but they need to be updated in order to make them efficiently applicable to large data sets.

D

Overview

Data stored in memory devices, in storage networks, on the Web, or in the Cloud must be protected against loss, accidental contamination, or deliberate adulteration. Data are valuable assets that can be lost to negligence or theft (for illicit use or to exchange for ransom). Over the years, many methods of data protection have been devised by researchers in the field of dependable and fault-tolerant computing (Jalote 1994; Parhami 2018), all of which entail introducing redundancy to make data robust and recoverable in the event of loss or corruption. As data assumes ever-more important roles in the proper functioning of systems that affect our daily lives, greater protection, often involving higher degrees of redundancy, becomes necessary. Yet, the age of big data (Hilbert and Gomez 2011) makes naive redundancy methods costlier and less convenient, given the exponentially growing data volume, which has seen aggregate data storage capacity in the world rise from exabytes in 1986 to zettabytes at the turn of the century (Hilbert and Gomez 2011).

Data Unavailability, Corruption, and Loss

As valuable resources, data assets are subject to theft (can be worse than loss, if the entity stealing it abuses the resource), unavailability (not as bad as loss or corruption but inconvenient nonetheless), corruption, or permanent loss. Theft for illicit use can be prevented by encryption, a topic that is outside the scope of this article (Hankerson et al. 2000).

Data is lost or becomes unavailable for use if the storage device or host site holding the data crashes or is otherwise unresponsive. Data unavailability can be avoided by keeping multiple copies of the data on different devices/sites or by means of distributed encoding.

The term “error” refers to any change of bit or symbol values from the original or intended values. Errors can be classified by their type (inversion, erasure, substitution, and so on) and extent (the number and positions of the bits/symbols affected). Accordingly, error codes can be single- or multiple-error detecting/correcting. In the case of multiple errors, they can be arbitrary or correlated, important examples of the latter type being unidirectional and burst errors.

The most immediate threats to data corruption are inadvertent changes to bit or symbol values due to hardware failures (during the storage or processing of data) or transmission errors (as the data is moved from one place to another). The corruption extent ranges from one bit-flip ($0 \rightarrow 1$ or $1 \rightarrow 0$) to complete erasure via all bits being changed to 0s, say. Data corruption can be dealt with in ways similar to data loss or unavailability. Corruption that is limited in scope can be handled by error codes applied to locally stored data. We thus devote two sections to error-detecting and error-correcting codes, before describing schemes capable of dealing with more widespread data corruption.

Error codes are means of representing data so as to detect corrupting changes or to fix the corrupted values automatically (Rao and Fujiwara 1989). Of course, detection by itself is inadequate for avoiding data loss. It must be combined with a data reconstruction or recovery scheme in order to regain the original data. The recovery strategy is often based on data replication, where backup copies of data are kept in a reasonably up-to-date manner for use during recovery.

Error codes are assessed by their extent of redundancy, the total number n of bits/symbols required to represent k data bits/symbols, with $r = n - k$ being the absolute redundancy and r/k constituting the redundancy ratio, and by their detection/correction capabilities (how many and what types of errors are detected or corrected).

Error-Detecting Codes

The simplest error-detecting codes (Wakerly 1978) are based on parity, that is, the evenness or oddness of the number of 1 s in a code consisting of 0s and 1s. A simple parity code has code distance of 2, meaning that any two code words are different in at least two positions, ensuring the detection of any single-bit error. Multiple parity bits can be used in a variety of ways, leading to a wide class of codes (Garrett 2004), including Hamming error-detecting and error-correcting codes (Hamming 1950).

Information flow in coded protection of data is typically as follows:

Input → Encode → Send → Store → Send →
Decode → Output

Such a coding framework protects the middle three stages of sending to memory, residence in memory, and sending to the final destination. If data processing is also included, it corresponds to a feedback path from the decode stage back to the encode stage through a processing unit. In other words, the redundant information is stripped, and data in its original nonredundant form is subjected to processing, with the result re-encoded before being stored. Communication application of coding corresponds to the linear path from input to output depicted above, while computing applications also include the feedback path just discussed.

Protection against storage errors has a lot in common with schemes of circumventing transmission errors, so the same codes are often applicable. However, there are specific codes that can exploit the data storage pattern, say, on two-dimensional recording surfaces, to provide greater protection at the same or lower cost (e.g., Parhami and Avizienis 1973).

With codes that are closed under processing, the feedback path can be shortened to span only the middle three, rather than five, stages, thus obviating the need for stripping and reinstating the code. Processing coded information directly provides stronger protection against accidental or hardware-induced data corruption. For example, arithmetic error-detecting codes (Avizienis 1971;

Avizienis 1973; Garner 1966) allow direct addition and subtraction of coded numbers, yielding coded results. Multiplication with arithmetic-coded operands is a bit more involved but still feasible, whereas division is quite difficult, thus limiting the scope of applicability.

Popular examples of error-detecting codes include checksum codes (McAuley 1994); weight-based and Berger codes (Berger 1961; Knuth 1986); cyclic codes (Peterson and Brown 1961); linear codes (Sklar and Harris 2004), which include Hamming codes (Hamming 1950) as special cases, balanced codes (Knuth 1986); arithmetic codes (Avizienis 1971); and many other types developed for specific applications.

Error-Correcting Codes

Error-correcting codes (Arazi 1987, Peterson and Weldon 1972) are used extensively in storing data on disk memory, whose raw reliability would be dismal without the detecting and correcting capabilities of such codes (Petascale Data Storage Institute 2012; Schroeder and Gibson 2007), as well as in many other application contexts. The use of such codes within individual data blocks is often augmented by inter-block redundancy schemes that provide an extra layer of protection in what is known as RAID architecture, acronym for redundant array of independent disks (Chen et al. 1994; Feng et al. 2005), or redundant disk array, for short.

Popular examples of error-correcting codes in practical use include cyclic codes (Lin and Costello 2004), which include the widely used cyclic redundancy check or CRC codes as special cases; linear codes (Sklar and Harris 2004), which include Hamming codes (Hamming 1950) as special cases; Reed-Solomon codes (Feng et al. 2005; Reed and Solomon 1960); BCH codes (Pless 1998); and many other types developed for specific applications. Turbo codes (Benedetto and Montorsi 1996) and low-density parity-check or LDPC codes (Gallager 1962) are examples of widely studied and used modern codes that achieve very low redundancy ratios.

The theory of error-detecting and error-correcting codes is quite advanced, and progress is still being made (Guruswami and Rudra 2009). What remains missing is a methodology for choosing an optimal or near-optimal code for a given set of application requirements.

D

Replication Codes

Replication codes are the conceptually simplest codes but imply high redundancy (100% in the case of duplication, 200% for triplication). Their advantages include the fact that any unrestricted error in one copy is detectable (duplication) or correctable (triplication). Such detection and correction capabilities require that data copies be stored on distributed nodes whose failures are independent of each other (Jalote 1994).

In addition to allowing error detection or correction, replication facilitates access to data by improving availability and access latency. For example, with two copies of a particular piece of data stored on two different computers, extreme slowness or even crashing of one computer will not hinder access to the data. The prevailing nomenclature here is that we have a primary site for each piece of data and one or more backup sites (Budhiraja et al. 1993; Guerraoui and Schiper 1997; Jalote 1994). In addition to high redundancy in such primary-backup schemes, a key challenge is to keep the multiple copies up-to-date and consistent with each other (Dullmann et al. 2001).

Data Dispersion

An alternative to replication is a scheme known as data dispersion (Iyengar et al. 1998; Rabin 1989), originally devised by Michael O. Rabin, which entails much less redundancy but requires some computation to reconstruct the data.

Let us present this method through an example. Consider the three numbers a , b , and c as the data of interest to be protected. Instead of storing the three numbers, we might store the four values $f(1)$, $f(2)$, $f(3)$, and $f(4)$, where f is the polynomial function $f(x) = ax^2 + bx + c$. We

can lose any one of the four f values and still be able to derive the original data values a , b , and c from the three remaining f values by solving a system of three linear equations. This example, which entails 33% redundancy to protect against loss of any single piece of data (compared with 100% for duplication), is readily understood, but it is not a particularly efficient way of dispersing data.

Data dispersion is actually a kind of encoding (Feng et al. 2005), where k pieces of data (say, k numbers) are encoded into n pieces (with $n > k$), in such a way that any k pieces suffice to derive the original data. Besides being useful for data protection, the method has security and load-balancing applications. In security, imagine devising a scheme so that at least three officers of a bank must cooperate to gain access to a safe, whereas no two officers can gain access. Such an arrangement is sometimes referred to as “secret sharing” (Beimel 2011). In load balancing, consider that the k pieces retrieved for reconstructing the original data can come from any of the stakeholders, thus making it possible to avoid congested sites. In such a case, data dispersion can be viewed as a performance-enhancing strategy, as well as a reliability improvement method.

Future Directions

The field of dependable computing has advanced for several decades to provide more reliable processing, storage, and communication resources. It is well known that reliability is a weakest-link phenomenon, in the sense that any unprotected or weakly protected part of the system can doom the entire system. The exponential reliability law (Parhami 2018) suggests that reliability declines exponentially with an increase in the number of system parts. In the context of big data, all aspects of the system (processing, storage, and communication) will expand in complexity, challenging the system designer to seek more advanced methods of reliability assurance.

Design of computer systems and their attendant attributes in light of new application domains and technological developments in the

twenty-first century is under review by many researchers (Chen and Zhang 2014; Hu et al. 2014; Stanford University 2012). Since resources and services for big data will be provided through the Cloud, directions of cloud computing (Armburst et al. 2010) and associated hardware acceleration mechanisms become relevant to our discussion here (Caulfield et al. 2016).

A promising direction is provided by network coding (Dimakis et al. 2011). This new field essentially expands on the idea of data dispersion by considering more general data distribution schemes while also taking the impact of “repair traffic” on system performance into account.

Cross-References

- ▶ [Data Longevity and Compatibility](#)
- ▶ [Hardware Reliability Requirements](#)
- ▶ [Parallel Processing with Big Data](#)

References

- Arazi B (1987) A commonsense approach to the theory of error-correcting codes. MIT Press, Cambridge, MA
- Armburst M et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
- Avizienis A (1971) Arithmetic error codes: cost and effectiveness studies for application in digital system design. IEEE Trans Comput 20(11):1322–1331
- Avizienis A (1973) Arithmetic algorithms for error-coded operands. IEEE Trans Comput 22(6):567–572
- Beimel A (2011) Secret-sharing schemes: a survey. In: Proceedings of international conference coding and cryptology, Springer LNCS no. 6639, Berlin, pp 11–46
- Benedetto S, Montorsi G (1996) Unveiling turbo codes: some results on parallel concatenated coding schemes. IEEE Trans Inf Theory 42(2):409–428
- Berger JM (1961) A note on error detection codes for asymmetric channels. Inf Control 4:68–73
- Budhiraja N, Marzullo K, Schneider FB, Toueg S (1993) The primary-backup approach. Distrib Syst 2:199–216
- Caulfield AM, et al (2016) A cloud-scale acceleration architecture. In: Proceedings of 49th IEEE/ACM international symposium on microarchitecture, Taipei, Taiwan, pp 1–13
- Chen CLP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347
- Chen PM, Lee EK, Gibson GA, Katz RH, Patterson DA (1994) RAID: high-performance reliable secondary storage. ACM Comput Surv 26(2):145–185

- Dimakis AG, Ramachandran K, Wu Y, Suh C (2011) A survey on network codes for distributed storage. *Proc IEEE* 99(3):476–489
- Dullmann D et al (2001) Models for replica synchronization and consistency in a data grid. In: Proceedings of 10th IEEE international symposium on high performance distributed computing, San Francisco, CA, pp 67–75
- Feng G-L, Deng RH, Bao F, Shen J-C (2005) New efficient MDS array codes for RAID – part I: Reed-Solomon-like codes for tolerating three disk failures. *IEEE Trans Comput* 54(9):1071–1080; Part II: Rabin-like codes for tolerating multiple (≥ 4) disk failures. *IEEE Trans Comput* 54(12):1473–1483
- Gallager R (1962) Low-density parity-check codes. *IRE Trans Inf Theory* 8(1):21–28
- Garner HL (1966) Error codes for arithmetic operations. *IEEE Trans Electron Comput* 5:763–770
- Garrett P (2004) The mathematics of coding theory. Prentice Hall, Upper Saddle River
- Guerraoui R, Schiper A (1997) Software-based replication for fault tolerance. *IEEE Comput* 30(4):68–74
- Guruswami V, Rudra A (2009) Error correction up to the information-theoretic limit. *Commun ACM* 52(3):87–95
- Hamming RW (1950) Error detecting and error correcting codes. *Bell Labs Tech J* 29(2):147–160
- Hankerson R et al (2000) Coding theory and cryptography: the essentials. Marcel Dekker, New York
- Hilbert M, Gomez P (2011) The World's technological capacity to store, communicate, and compute information. *Science* 332:60–65
- Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics; a technology tutorial. *IEEE Access* 2:652–687
- Iyengar A, Cahn R, Garay JA, Jutla C (1998) Design and implementation of a secure distributed data repository. IBM Thomas J. Watson Research Division, Yorktown Heights
- Jalote P (1994) Fault tolerance in distributed systems. Prentice Hall, Englewood Cliffs
- Knuth DE (1986) Efficient balanced codes. *IEEE Trans Inf Theory* 32(1):51–53
- Lin S, Costello DJ (2004) Error control coding, vol 2. Prentice Hall, Upper Saddle River
- McAuley AJ (1994) Weighted sum codes for error detection and their comparison with existing codes. *IEEE/ACM Trans Networking* 2(1):16–22
- Parhami B (2018) Dependable computing: a multi-level approach. Draft of book manuscript, available on-line at: http://www.ece.ucsb.edu/~parhami/text_dep_comp.htm
- Parhami B, Avizienis A (1973) Detection of storage errors in mass memories using arithmetic error codes. *IEEE Trans Comput* 27(4):302–308
- Petascale Data Storage Institute (2012) Analyzing failure data. Project Web site: <http://www.pdl.cmu.edu/PDSI/FailureData/index.html>
- Peterson WW, Brown DT (1961) Cyclic codes for error detection. *Proc IRE* 49(1):228–235
- Peterson WW, Weldon EJ Jr (1972) Error-correcting codes, 2nd edn. MIT Press, Cambridge, MA
- Pless V (1998) Bose-Chaudhuri-Hocquenghem (BCH) codes. In: Introduction to the theory of error-correcting codes, 3rd edn. Wiley, New York, pp 109–222
- Rabin M (1989) Efficient dispersal of information for security, load balancing, and fault tolerance. *J ACM* 36(2):335–348
- Rao TRN, Fujiwara E (1989) Error-control coding for computer systems. Prentice Hall, Upper Saddle River, NJ
- Reed I, Solomon G (1960) Polynomial codes over certain finite fields. *SIAM J Appl Math* 8:300–304
- Schroeder B, Gibson GA (2007) Understanding disk failure rates: what does an MTTF of 1,000,000 hours mean to you? *ACM Trans Storage* 3(3):Article 8, 31 pp
- Sklar B, Harris FJ (2004) The ABCs of linear block codes. *IEEE Signal Process* 21(4):14–35
- Stanford University (2012) 21st century computer architecture: a community white paper. On-line: http://csl.stanford.edu/~christos/publications/2012.21st_centuryarchitecture.whitepaper.pdf
- Wakerly JF (1978) Error detecting codes, self-checking circuits and applications. North Holland, New York

D

Data Retention

- Computing the Cost of Compressed Data

Data Storage Replication

- Geo-replication Models

Data Structures for Graphs

- (Web/Social) Graph Compression

Data Validation

- Data Quality and Data Cleansing of Semantic Data

Data Warehouse Benchmark

► TPC-H

Data Wrangling

Jeffrey Heer¹, Joseph M. Hellerstein², and Sean Kandel³

¹University of Washington, Seattle, WA, USA

²University of California, Berkeley, Berkeley, CA, USA

³Trifacta Inc., San Francisco, CA, USA

Synonyms

Data preparation

Definitions

Data wrangling is the process of profiling and transforming datasets to ensure they are actionable for a set of analysis tasks. One central goal is to make data *usable*: to put data in a form that can be parsed and manipulated by analysis tools. Another goal is to ensure that data is *responsive* to the intended analyses: that the data contain the necessary information, at an acceptable level of description and correctness, to support successful modeling and decision-making.

Overview

Despite significant advances in technologies for data management and analysis, it remains time-consuming to inspect a dataset and mold it to a form that allows meaningful analysis to begin. Analysts must regularly restructure data to make it palatable to databases, statistics packages, and visualization tools. To improve data quality, analysts must also identify and address issues such as misspellings, missing data, unresolved duplicates, and outliers.

Data wrangling is the process of inspecting and transforming data to ensure it is actionable for analysis, and often consumes the lion's share of effort within data analysis efforts. Wrangling tasks range from early-stage parsing and formatting to data quality assessment and cleaning and, to data integration and distillation. While each of these topics has their own rich literatures, research on data wrangling takes a distinctly user-centered focus: how might we empower analysts to effectively orchestrate data profiling and transformation methods to ensure data is usable and responsive to analysis goals?

Interviews with data professionals reveal that a majority of their time is spent in data wrangling tasks, which can be relatively unsophisticated and repetitive, yet persistently tricky and time-consuming (Kandel et al. 2012a). The most-cited bottleneck arises in manually preparing the various sources of data required for each distinct use case. Others estimate that data cleaning is responsible for up to 80% of the development time and cost in data warehousing projects (Dasu and Johnson 2003).

This effort arises because data wrangling is an inherently *iterative* and *interactive* process. Nuanced judgments requiring domain expertise are sometimes needed, beyond the scope of analysis algorithms. For example, is an observed outlier a valid finding or an error to be discarded? Should records with missing data be kept as is, removed, or corrected via imputation (and if so, by which method)? In addition, as new questions arise and analysis goals evolve, the appropriate set of data transformations may correspondingly shift, requiring additional wrangling. Data wrangling is thus resistant to full automation. As a result, technologies for data wrangling often take the form of interactive systems, with a goal of balancing scalable, automated subroutines with human guidance and assessment.

The technical challenges of data wrangling come from the unbounded variety of inputs and outputs to the problem, which often necessitate custom work. Traditionally, data wrangling has required writing idiosyncratic scripts in programming languages such as Python and R or extensive manual editing using interactive tools such

as spreadsheets. This hurdle arguably discourages many people from working with data in the first place. The end result is that domain experts regularly spend more time manipulating data than they do exercising their speciality, while less technical audiences are needlessly excluded.

At heart, data wrangling is a domain-specific programming problem, with a strong focus on the structure, semantics, and statistical properties of data. This observation suggests that by making the specification of data transformation programs dramatically easier, we might remove drudgery for scarce technical workers and engage a far wider labor pool in time-consuming, data-centric tasks.

Key Research Findings

Data wrangling covers a range of tasks, including structuring (e.g., extracting fields from text, table reshaping), profiling (e.g., summary visual-

ization, outlier detection), standardization (e.g., entity resolution, error correction, imputation), enrichment (e.g., lookups, joins), and distillation (e.g., sampling, filtering, aggregation, windowing). Ideally, the outcome of wrangling is not simply data; it is an editable and auditable transcript of transformations coupled with a nuanced understanding of data organization and data quality issues.

D

Data Parsing and Structuring

A common first task for data wrangling is to *parse* and *structure* a new data source. As illustrated in Fig. 1, small datasets such as those found in spreadsheets may be primarily intended for manual inspection; restructuring such data allows it to be more easily visualized, modeled, or integrated with other datasets. Related challenges arise with larger datasets spanning gigabytes, terabytes, or more. For example, log files (e.g., of user interactions or system operations) often use idiosyncratic text formats or semi-structured

Data Wrangling, Fig. 1

An example of wrangling a US government personnel dataset. Given spreadsheet-formatted data as input, as set of transformations are applied to make the data more amenable to computational use

1. Input data table

Bureau of I.A.	
Regional Director	Numbers
Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

2. Delete first two rows

Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

3. Split column 2 on ‘:’

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

4. Delete empty rows

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

5. Fill column 1 from above

Niles C.	Tel	(800)645-8397
Niles C.	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
Jean H.	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
Frank K.	Fax	(615)564-6701

6. Pivot column 2 on column 3

Niles C.	Tel	Fax
Jean H.	Tel	Fax
Frank K.	Tel	Fax

formats that require transformation prior to integrating with relational databases. In many cases, data resides in a text-based format that must be mapped to a corresponding tabular representation.

Data parsing involves first identifying the data format. For text-based formats one must segment the text into records and attributes. Other data may use a semi-structured format (XML, JSON) that may include nested sub-records. At this stage, one often performs *type induction* to identify the data types of individual attributes. This parsing stage can be automated for many common formats but may require interactive assistance for idiosyncratic or inconsistent encodings.

Once parsed, data must sometimes be restructured. A common goal is to ensure the data uses a *tidy* format (Wickham 2014): a tabular format in which each row corresponds to a single observation and each column represents a semantically consistent variable with a single data type. In terms of database theory, a “tidy” format is closely related to a relation in first normal form (1NF) (Codd 1971b). (Normal forms beyond first normal form (second normal form, etc.) are often less desirable for analysis purposes: one might wish to denormalize data (e.g., by joining relations with primary-foreign key relationships) in order to more conveniently perform analysis over a single table.)

Common operations for string-typed data include *splitting* columns (e.g., based on delimiters), pattern-based *extraction* of substrings into new columns, and *merging* columns together. In the case of semi-structured data, *unnesting* of sub-record attributes to flat columns is often necessary. In addition, reshaping operations may be needed to ensure a tidy organization. Consider an example table containing two columns – a string key and a numeric measure – where each unique key indicates a different type of measurement. This organization violates a tidy format, as semantically different measures are mixed within a column. This situation can be remedied via a *pivot* operation that reshapes the data into new columns, one for each unique key value (as in step 6 of Fig. 1). Conversely, spreadsheet data often

contains cross-tabulations by category or date, in which semantically similar measures are spread over multiple columns. These cross-tabulations can be mapped to a tidy format via an *unpivot* operation (also referred to as a *fold* or *melt* operation).

Data Profiling

Once data is suitably formatted, one can begin assessing the shape and structure of the data to guide further wrangling. The goal of *data profiling* is to gain a basic understanding of data content and assess potential data quality issues. Profiling typically involves consulting both summary statistics and data visualizations.

Visualization techniques are central to data profiling (Kandel et al. 2011a). Histograms are well-suited for showing the distributions of numerical data, though care must be taken regarding the choice of binning scheme. For categorical data, bar charts of occurrence counts, sorted by frequency, can depict top-K values. Other data types can benefit from specialized displays, for example, maps overlaid with symbols for geographic data and multiple plots showing various transformations for time-series data (e.g., histograms for month, day of month, day of week, etc.).

To scalably visualize relationships between variables, binned 2D histograms (a discretized version of scatterplots) can be used (Carr et al. 1987). In addition, interaction techniques such as brushing and linking (in which selections in one view are used to filter the data shown or highlighted in other views) can be applied to assess higher-dimensional relationships. Visualizations can also be augmented to aid identification of missing values, using either auxiliary visualizations (e.g., quality bars showing the proportions of valid, missing, and type-violating values, as in Kandel et al. 2011b) or by including missing values directly within a plot (Eaton et al. 2003) (e.g., including a specific category for null values).

An important goal during profiling is to account for data type violations, missing values, and outliers. Automatic identification of null values or values that do not match a column’s data

type is straightforward; once flagged they can be visually presented to a user for consideration and correction. Identification of missing records (i.e., entire table rows) is much more difficult but may be aided by looking for corresponding patterns in summary visualizations (e.g., gaps within a histogram of record timestamps).

Outliers can be identified within visualizations and using statistical routines (Hodge and Austin 2004; Hellerstein 2008). Univariate outliers can be flagged using basic summary statistics (e.g., distance from the interquartile range of numeric values), sometimes in conjunction with transformations (e.g., identifying outliers in string-typed data by analyzing the distribution of string lengths). Multivariate outlier detection is a more difficult problem, for which statistical measures such as Mahalanobis distance can be applied. Beyond numerical and categorical data, other types, including geographic and time-series data, can further benefit from specialized outlier detection techniques.

Once identified, missing or outlying values must often be interpreted in the context of other data columns (e.g., date and time of observation) and/or external knowledge (e.g., was there a power spike at that time that may have caused erroneous extreme observations?) to gauge whether the values are valid or potentially misleading. A user can then decide if corrective transformation is needed.

Data Standardization

In order to safeguard subsequent analysis, the data values within each column should be suitably *standardized*. Going beyond low-level physical types (e.g., floats, strings), many data wrangling tools support higher-level semantic types (e.g., zip codes, phone numbers) for ensuring values conform to a desired format. Data quality rules may be applied to ensure values correspond to either a strict set of legal values or satisfy a set of quality constraints. Users can perform corrective transformations to standardize values as needed. Generally speaking, any number of custom formulae may be applied, depending on the data type.

A common standardization task is *entity resolution* (or de-duplication), in which similar values representing the same entity are mapped to a single standardized value. For example, if we encounter similar names in a dataset, do they represent the same person? Entity resolution is a rich topic, with years of research into both automatic and interactive approaches (Elmagarmid et al. 2007). Common techniques include clustering to identify similar values (e.g., using string edit distance to group similar names) and blocking to aid disambiguation (consulting values in other columns to decide if similar values represent the same entity, e.g., requiring similar names to also come from the same geographic region), often followed by manual inspection (including crowdsourcing techniques to perform manual correction at scale (Stonebraker et al. 2013)).

Another standardization issue is the treatment of missing or outlier values. In the example of Fig. 1, a *fill* operation is used to copy string values into missing cells. In other cases, users may wish to remove missing data from consideration. For numerical data, interpolation (e.g., to fill in missing values over a time domain) or imputation (e.g., replacing a value with the mean value for a corresponding group) of missing values may be performed, depending on the downstream analysis needs. Similarly, outliers may be removed or replaced with an imputed value depending on the analysis goals; for example, some statistical modeling methods are highly sensitive to outliers.

Data Enrichment and Distillation

The above tasks have largely focused on wrangling a single dataset. However, in many cases a primary goal of wrangling is to combine multiple data sources in order to perform a more comprehensive analysis. This task is often referred to as *data enrichment* or *data blending* and is situated within the larger space of *data integration* tasks (Doan et al. 2012).

A basic form of enrichment is to *look up* additional details. To aid visualization and interpretation, one might use a lookup table to map from a product code to more human-friendly product name string. Such lookups are an example of data de-normalization to aid analysis. A related

operation is to perform geo-coding to retrieve the longitude-latitude coordinates for an address or zip code.

More complex operations include multi-table *join* and *union* operations. While common to database management systems, within data wrangling contexts joins and unions may involve datasets that were not originally designed to be used together. In the absence of explicit primary key and foreign key designations, key inference and schema mapping (Rahm and Bernstein 2001) may be needed to properly align tables. Moreover, additional transformation operations may be necessary to ensure compatible join keys or unionable columns.

Distillation is the task of summarizing or reducing a dataset prior to downstream analysis. For example, an analyst may wish to perform *filtering* or *sampling* to reduce the data or extract a subset appropriate for comparison (e.g., performing stratified sampling by group). *Aggregation* and *windowing* operations are means of summarizing data, often by subgroup. In some cases, aggregation is a necessary precursor to data integration, as datasets may use units of analysis with differing levels of granularity (e.g., US states versus US counties).

Organizational Deployment, Sharing, and Review

While data wrangling is an important consideration of an individual analyst's work, wrangling also plays a social role within an organization. A set of wrangling transformations may need to be applied repeatedly, scheduled to rerun at regular intervals as new data batches arrive. Once wrangled, data can be valuable to a number of various analysts or decision makers within an organization. Methods for sharing and discovery (e.g., *data cataloging*) can improve an organization's access to data and amortize wrangling effort.

The *provenance* of wrangled data is another central concern. Before using a dataset in their analyses, a user may wish to review what transformations were applied and ensure they are appropriate. An executive may wish to know what calculations were performed to derive a key per-

formance metric prior to making a dependent decision. Similarly, a compliance officer may wish to review how specific privacy-sensitive fields are subsequently used. Each of these tasks involves additionally tracking and visualizing the lineage of transformed data.

These needs imply that a critical output of the wrangling process is not simply transformed data, but rather a reusable, editable, and auditable transcript of wrangling operations. Not unlike the collaboration and management enabled by revision control tools for code, both wrangling programs and transformed data are organizational resources to be socially shared and monitored.

Examples of Application

To give a sense of the data wrangling process, Fig. 1 illustrates a scenario adapted from Guo et al. (2011), in which a user transforms spreadsheet data (step 1 within Fig. 1) into a dense relational format (step 6). This example begins with personnel data originally formatted for human reading and adapts it to a format more amenable to computational processing. The transformation process involves removing unneeded records (steps 2 and 4), splitting text fields into well-typed columns (step 3), interpolation of missing values (step 5), and finally reshaping the table via a pivot (or cross-tabulation) operation (step 6).

While involving only a small dataset, this example exhibits a number of common operations necessary for wrangling a dataset. In general, datasets of all sizes may require significant wrangling effort, whether to restructure the data for use by common tools, to identify and correct anomalies or discrepancies, or to adapt a dataset such that it can be integrated with other sources of data.

Future Directions for Research

Technologies for data wrangling aim to support one or more of the tasks described above. Wrangling tools often involve the careful integration of automatic algorithms (e.g., type inference,

transform inference, program synthesis) within interactive systems (e.g., an interactive console or graphical user interface) that enable users to inspect data and orchestrate transformations. Future work areas include (1) improved languages and runtimes for wrangling operations, (2) new user interface designs for data profiling and interactive transformation, and (3) novel inferential procedures for simplifying and scaling human guidance of the wrangling process.

Domain-Specific Languages for Data Wrangling

Data wrangling is, at its core, a programming problem: what transformations should be applied and in what order? A popular approach is thus to craft *domain-specific languages* (DSLs) that support wrangling tasks. Standard database query languages such as SQL support a number of relevant wrangling tasks (distillation, enrichment, etc.) but are less well-suited for messy data with type violations or requiring operations that transform between metadata and data (e.g., unpivot operations that map column names into data). The SchemaSQL (Lakshmanan et al. 2001) language generalizes SQL to include such second-order operations. The PADS Fisher and Walker (2011) and AJAX Galhardas et al. (2000) projects also introduce custom DSLs for data transformation. Within the R programming language for computational statistics, *dplyr* and *tidyverse* (Wickham 2014) are embedded DSLs supporting transformation and restructuring tasks. DSLs also underlie a number of interactive tools, such as Potter’s Wheel (Raman and Hellerstein 2001) and Wrangler (Kandel et al. 2011b; Guo et al. 2011). In addition to providing a succinct high-level syntax, a DSL can support compilation to alternative runtimes (e.g., computing clusters or database management systems) to enable scalable execution of transformation scripts.

Graphical Interfaces for Data Wrangling

While valuable, direct use of domain-specific languages primarily benefits programmers, as opposed to a larger swath of domain experts knowledgeable about their data but not necessarily skilled in programming. In response,

a number of graphical interfaces have been devised for supporting data wrangling. Some projects focus on new interaction techniques for a specific wrangling task, such as tools for deduplication (Sarawagi and Bhamidipaty 2002; Kang et al. 2008), schema matching (Robertson et al. 2005; Chiticariu et al. 2008), or data profiling (Dasu et al. 2002; Kandel et al. 2012b). Future research projects could develop methods for core wrangling tasks, including improved entity resolution, standardization, and analysis of data lineage.

Other projects focus on more complete environments for data wrangling. The Potter’s Wheel (Raman and Hellerstein 2001) system is an early example, providing both a spreadsheet-like table representation and menu-driven commands to perform transformations. Subsequent tools include Refine (Huynh and Mazzocchi 2010) and Wrangler (Kandel et al. 2011b; Guo et al. 2011) (later commercialized by Trifacta), which add richer support for direct manipulation and visualization.

Both the Potter’s Wheel (Raman and Hellerstein 2001) and Wrangler (Kandel et al. 2011b; Guo et al. 2011) systems map interactive operations to statements in an underlying DSL, enabling those tools to output transformation programs that can be cross-compiled to scalable runtime environments. For example, by interactively wrangling a sample of data, these tools can then generate programs that can be applied to transform terabytes (or more) of data. However, users of graphical interfaces are accustomed to directly manipulating the state of a document or dataset, not a more abstract set of (reusable) operations. Future work might explore and evaluate improved methods for representing and manipulating the *wrangling programs* created by such tools.

Inferential Methods to Accelerate Wrangling

A central feature to many successful wrangling tools is a judicious balance of user interaction and automated algorithms. Examples of useful inference methods include outlier detection, type inference (to determine column data types), join

key inference (to identify primary-foreign key relationships among multiple tables), and union inference (to determine appropriate column assignments – or lack thereof – between two or more tables). Each of these aids identification of important data characteristics and the formulation of appropriate wrangling operations (e.g., as statements in a wrangling DSL). An interactive wrangling tool must run the appropriate analysis (perhaps in response to user-initiated commands, such as selecting tables to join) and then surface the results in an actionable manner.

A number of projects have explored interactive methods, such as programming by example, to couple automatic inference with human guidance. The work of Gulwani and collaborators applies program synthesis methods to learn text extraction (Gulwani 2011) and table restructuring (Harris and Gulwani 2011) procedures from input-output examples; some of these techniques have been incorporated into Microsoft Excel.

Kandel et al.'s Wrangler (2011b) takes a different approach, performing search over the space of possible transformation operations, informed by user selections of data table rows, columns, or cell contents. Wrangler returns and visualizes potential transformations from which the user can choose (or modify). As users transform their data, the commercial Trifacta Wrangler product additionally uses automatic visualization routines to generate charts to aid data profiling. The goal of this *predictive interaction* (Heer et al. 2015) loop is recast the task of writing a DSL program into one of visually guiding and refining a transformation recommender system. Many challenges remain in perfecting mixed-initiative (Horvitz 1999) interaction techniques that use inferential methods to speed and scale wrangling, while preserving effective human assessment and guidance of the wrangling process.

Cross-References

- ▶ [Data Cleaning](#)
- ▶ [Data Integration](#)
- ▶ [Data Profiling](#)

- ▶ [Data Quality and Data Cleansing of Semantic Data](#)
- ▶ [ETL](#)
- ▶ [Large-Scale Entity Resolution](#)

References

- Carr DB, Littlefield RJ, Nicholson W, Littlefield J (1987) Scatterplot matrix techniques for large N. *J Am Stat Assoc* 82(398):424–436
- Chiticariu L, Kolaitis PG, Popa L (2008) Interactive generation of integrated schemas. In: ACM SIGMOD, pp 833–846
- Codd EF (1971b) Further normalization of the data base relational model. In: Courant computer science symposia 6, Data base systems, (New York, May 24–25) pp 33–64, Prentice-Hall
- Dasu T, Johnson T (2003) Exploratory data mining and data cleaning. Wiley, New York
- Dasu T, Johnson T, Muthukrishnan S, Shkapenyuk V (2002) Mining database structure; or, how to build a data quality browser. In: ACM SIGMOD, pp 240–251
- Doan A, Halevy A, Ives Z (2012) Principles of data integration. Elsevier, Amsterdam
- Eaton C, Plaisant C, Drizd T (2003) The challenge of missing and uncertain data. In: Proceedings of the IEEE visualization, p 100
- Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: a survey. *IEEE TKDE* 19(1):1–16
- Fisher K, Walker D (2011) The PADS project: an overview. In: International conference on database theory, Mar 2011
- Galhardas H, Florescu D, Shasha D, Simon E (2000) AJAX: an extensible data cleaning tool. In: ACM SIGMOD, p 590
- Gulwani S (2011) Automating string processing in spreadsheets using input-output examples. In: ACM POPL, pp 317–330
- Guo PJ, Kandel S, Hellerstein J, Heer J (2011) Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In: ACM user interface software & technology (UIST)
- Harris W, Gulwani S (2011) Spreadsheet table transformations from examples. In: ACM PLDI
- Heer J, Hellerstein JM, Kandel S (2015) Predictive interaction for data transformation. In: CIDR
- Hellerstein JM (2008) Quantitative data cleaning for large databases. White Paper, United Nations Economic Commission for Europe
- Hodge V, Austin J (2004) A survey of outlier detection methodologies. *Artif Intell Rev* 22(2):85–126
- Horvitz E (1999) Principles of mixed-initiative user interfaces. In: ACM CHI, pp 159–166
- Huynh D, Mazzocchi S (2010) Google refine. <http://code.google.com/p/google-refine/>

- Kang H, Getoor L, Shneiderman B, Bilgic M, Licamele L (2008) Interactive entity resolution in relational data: a visual analytic tool and its evaluation. *IEEE TVCG* 14(5):999–1014
- Kandel S, Heer J, Plaisant C, Kennedy J, van Ham F, Riche NH, Weaver C, Lee B, Brodbeck D, Buono P (2011a) Research directions in data wrangling: visualizations and transformations for usable and credible data. *Inf Vis* 10(4):271–288
- Kandel S, Paepcke A, Hellerstein J, Heer J (2011b) Wangler: interactive visual specification of data transformation scripts. In: ACM human factors in computing systems (CHI)
- Kandel S, Paepcke A, Hellerstein J, Heer J (2012a) Enterprise data analysis and visualization: an interview study. In: IEEE visual analytics science & technology (VAST)
- Kandel S, Parikh R, Paepcke A, Hellerstein J, Heer J (2012b) Profiler: integrated statistical analysis and visualization for data quality assessment. In: Advanced visual interfaces
- Lakshmanan LVS, Sadri F, Subramanian SN (2001) SchemaSQL: an extension to SQL for multidatabase interoperability. *ACM Trans Database Syst* 26(4): 476–519
- Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. *VLDB J* 10:334–350
- Raman V, Hellerstein JM (2001) Potter's wheel: an interactive data cleaning system. In: VLDB, pp 381–390
- Robertson GG, Czerwinski MP, Churchill JE (2005) Visualization of mappings between schemas. In: ACM CHI, pp 431–439
- Sarawagi S, Bhadridipaty A (2002) Interactive deduplication using active learning. In: ACM SIGKDD
- Stonebraker M, Bruckner D, Ilyas IF, Beskales G, Cherniack M, Zdonik SB, Pagan A, Xu S (2013) Data curation at scale: the data tamer system. In: CIDR
- Wickham H (2014) Tidy data. *J Stat Softw* 59(10):1–23

Database Consistency Models

Marc Shapiro¹ and Pierre Sutra²

¹Sorbonne Université, LIP6 and INRIA, Paris, France

²Télécom Sud Paris, Évry, France

Synonyms

Consistency criterion; Consistency model; Data consistency; Isolation level

The distributed systems and database communities use the same word, consistency, with differ-

ent meanings. Within this entry, and following the usage of the distributed algorithms community, “consistency” refers to the observable behavior of a data store.

In the database community, roughly the same concept is called “isolation,” whereas the term “consistency” refers to the property that application code is sequentially safe (the C in ACID).

D

Definitions

A data store allows application processes to put and get data from a shared memory. In general, a data store cannot be modeled as a strictly sequential process. Applications observe non-sequential behaviors, called anomalies. The set of possible behaviors, and conversely of possible anomalies, constitutes the *consistency model* of the data store.

Overview

Background

A *data store*, or database system, is a persistent shared memory space, where different client application processes can store data items. To ensure scalability and dependability, a modern data store distributes and replicates its data across clusters of *servers* running in parallel. This approach supports high throughput by spreading the load, low latency by parallelizing requests, and fault tolerance by replicating demand processing; system capacity increases by simply adding more servers (scale out).

Ideally, from the application perspective, data replication and distribution should be transparent. Read and update operations on data items would appear to execute as in a single sequential thread; reading a data item would return exactly the last value written to it in real time; and each application transaction (a grouping of operations) would take effect atomically (all at once). This ideal behavior is called *strict serializability*, noted SSER (Papadimitriou 1979).

In practice, exposing some of the internal parallelism to clients enables better perfor-

mance. More fundamentally, SSER requires assumptions that are unrealistic at large scale, such as absence of network partitions (see section “[Fundamental Results](#)” hereafter). Therefore, data store design faces a *fundamental tension* between providing a strictly serializable behavior on the one hand, versus availability and performance on the other. This explains why large-scale data stores hardly ever provide the SSER model, with the notable exception of Spanner (Corbett et al. 2012).

Consistency Models

Informally, a *consistency model* defines what an application can observe about the updates and reads of its data store. When the observed values of data differ from strict serializability, this is called an *anomaly*. Examples of anomalies include *divergence*, where concurrent reads of the same item persistently return different values; *causality violation*, where updates are observed out of order; *dirty reads*, where a read observes the effect of a transaction that has not terminated; or *lost updates*, where the effect of an update is lost. The more anomalies allowed by a store, the *weaker* its consistency model. The *strongest* model is (by definition) SSER, the baseline against which other models are compared.

More formally, a consistency model is defined by the history of updates and reads that clients

can observe. A model is weaker than another if it allows more histories.

An absolute definition of “strong” and “weak consistency” is open to debate. For the purpose of this entry, we say that a consistency model is *strong* when it has consensus power, i.e., any number of failure-prone processes can reach agreement on some value by communicating through data items in the store. If this is not possible, then the consistency model is said *weak*.

In a strong model, updates are totally ordered. Some well-known strong models include SSER, serializability (SER), or snapshot isolation (SI). Weak models admit concurrent updates to the same data item and include causal consistency (CC), strong eventual consistency (SEC), and eventual consistency (EC) (see Table 1).

Key Research Findings

Basic Concepts

A *data store* is a logically shared memory space where different application processes store, update, and retrieve data *items*. An item can be very basic, such as a register with read/write operations, or a more complex structure, e.g., a table or a file system directory. An application process executes *operations* on the data store through the help of an *API*. When a process

Database Consistency Models, Table 1 Models and source references

Acronym	Full name	Reference
EC	Eventual Consistency	Ladin et al. (1990)
SEC	Strong Eventual Consistency	Shapiro et al. (2011)
CM	Client monotonicity	Terry et al. (1994)
CS	Causal Snapshot	Chan and Gray (1985)
CC	Causal Consistency	Ahamad et al. (1995)
Causal HAT	Causal Highly-Av. Txn.	Bailis et al. (2013)
LIN	Linearisability	Herlihy and Wing (1990)
NMSI	Non-Monotonic SI	Saeida Ardekani et al. (2013a)
PSI	Parallel SI	Sovran et al. (2011)
RC	Read Committed	Berenson et al. (1995)
SC	Sequential Consistency	Lamport (1979)
SER	Serialisability	Gray and Reuter (1993)
SI	Snapshot Isolation	Berenson et al. (1995)
SSER	Strict Serialisability	Papadimitriou (1979)
SSI	Strong Snapshot Isolation	Daudjee and Salem (2006)

invokes an operation, it executes a remote call to an appropriate *endpoint* of the data store. In return, it receives a *response value*. A common example of this mechanism is a POST request to an HTTP endpoint.

An application consists of *transactions*. A transaction consists of any number of reads and updates to the data store. It is terminated either by an *abort*, whereby its writes have no effect, or by a *commit*, whereby writes modify the store. In what follows, we consider only committed transactions. The transaction groups together low-level storage operations into a higher-level abstraction, with properties that help developers reason about application behavior.

The properties of transactions are often summarized as ACID: All-or-Nothing, (individual) Correctness, Isolation, and Durability.

All-or-Nothing ensures that, at any point in time, either all of a transaction's writes are in the store or none of them is. This guarantee is essential in order to support common data invariants such as equality or complementarity between two data items. *Individual Correctness* is the requirement that each of the application's transactions individually transitions the database from a safe state (i.e., where some application-specific integrity invariants hold over the data) to another safe state. *Durability* means that all later transactions will observe the effect of this transaction after it commits. A, C, and D are essential features of any transactional system and will be taken for granted in the rest of this entry.

The I property, *Isolation* characterizes the absence of interference between transactions. Transactions are isolated if one cannot interfere with the other, regardless of whether they execute in parallel or not.

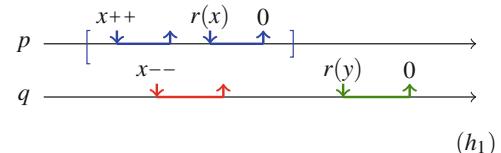
Taken together, the ACID properties provide the *serializability* model, a program semantics in which a transaction executes as if it was the only one accessing the database. This model restricts the allowable interactions among concurrent transactions, such that each one produces results indistinguishable from the same transactions running one after the other. As a consequence, the code for a transaction lives in the simple, familiar sequential programming world, and the devel-

oper can reason only about computations that start with the final results of other transactions. Serializability allows concurrent operations to access the data store and still produce predictable, reproducible results.

D

Definitions

A *history* is a sequence of invocations and responses of operations on the data items by the application processes. It is commonly represented with timelines. For instance, in history h_1 below, processes p and q access a data store that contains two counters (x and y).



(h_1)

Operations ($z++$) and ($z--$) respectively increment and decrement counter z by 1. To fetch the content of z , a process calls $r(z)$. A counter is initialized to 0. The start and the end of a transaction are marked using brackets, e.g., transaction $T_1 = (x++) . r(x)$ in history h_1 . When the transaction contains a single operation, the brackets are omitted for clarity.

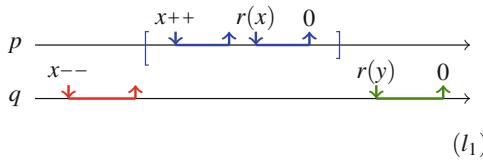
As pointed above, we assume in this entry that all the transactions are committed in a history. Thus every invocation has a matching response. More involved models exist, e.g., when considering a transactional memory (Guerraoui and Kapalka 2008).

A history induces a real-time order between transactions (denoted \prec_h). This order holds between two transactions T and T' when the response of the last operation in T precedes in h the invocation of the first operation in T' . A history also induces a per-process order that corresponds to the order in which processes invoke their transactions. For instance in h_1 , transaction $T_2 = (x--)$ precedes transaction $T_3 = r(y)$ at process q . This relation together with $(T_1 \prec_{h_1} T_3)$ fully defines the real-time order in history h_1 .

Histories have various properties according to the way invocations and responses interleave.

Two transactions are concurrent in a history h when they are not ordered by the relation \prec_h . A history h is *sequential* when no two transactions in h are concurrent. A sequential history is *legal* when it respects the sequential specification of each object. Two histories h and h' are *equivalent* when they contain the same set of events (invocations and responses).

A consistency model defines the histories that are allowed by the data store. In particular, serializability (SER) requires that every history h is equivalent to some sequential and legal history l . For instance, history h_1 is serializable, since it is equivalent to the history l_1 below.



In addition, if the equivalent sequential history preserves the real-time order between transaction, history h is said *strictly serializable* (SSER) (Papadimitriou 1979). This is the case of h_1 since in l_1 the relations $(T_2 \prec_{h_1} T_3)$ and $(T_1 \prec_{h_1} T_3)$ also hold.

When each transaction contains a single operation, SSER boils down to linearizability (LIN) (Herlihy and Wing 1990). The data store ensures sequential consistency (SC) (Lamport 1979) when each transaction contains a single operation and only the per-process order is kept in the equivalent sequential history.

The above consistency models (SER, SSER, LIN, and SC) are strong, as they allow the client application processes to reach consensus. To see this, observe that processes may agree as follows: The processes share a FIFO queue L in the data store. To reach consensus, each process enqueues some value in L which corresponds to a proposal to agree upon. Then, each process chooses the first proposal that appears in L . The equivalence with a sequential history implies that all the application processes pick the same value.

Conversely, processes cannot reach consensus if the consistency model is *weak*. A widespread

model in this category is Eventual Consistency (EC) (Vogels 2008), used, for instance, in the Simple Storage Service (Murty 2008). EC requires that, if clients cease submitting transactions, they eventually observe the same state of the data store. This eventually-stable state may include part (or all) the transactions executed by the clients. Under EC, processes may repeatedly observe updates in different orders. For example, if the above list L is EC, each process may see its update applied first on L until it decides, preventing agreement. In fact, EC is too weak to allow asynchronous failure-prone processes to reach an agreement (Attiya et al. 2017).

Fundamental Results

In the most general model of computation, replicas are asynchronous. In this model, and under the hypothesis that a majority of them are correct, it is possible to emulate a linearizable shared memory (Attiya et al. 1990). This number of correct replicas is tight. In particular, if any majority of the replicas may fail, the emulation does not work (Delporte-Gallet et al. 2004).

The above result implies that, even for a very basic distributed service, such as a register, it is not possible to be at the same time consistent, available, and tolerant to partition. This result is known as the CAP Theorem (Gilbert and Lynch 2002), which proves that it is not possible to provide all the following desirable features at the same time: (C) strong Consistency, even for a register; (A) Availability, responding to every client request; and (P) tolerate network Partition or arbitrary messages loss.

A second fundamental result, known as FLP, is the impossibility to reach consensus deterministically in presence of crash failures (Fischer et al. 1985). FLP is true even if all the processes but one are correct.

As pointed above, a majority of correct processes may emulate a shared memory. Thus, the FLP impossibility result indicates that a shared memory is not sufficient to reach consensus. In fact, solving consensus requires the additional ability to elect a leader among the correct processes (Chandra et al. 1996).

Data stores that support transactions on more than one data item are subject to additional impossibility results. For instance, an appealing property is genuine partial replication (GPR) (Schiper et al. 2010), a form of disjoint-access parallelism (Israeli and Rappoport 1994). Under GPR, transactions that access disjoint items do not contend in the data store. GPR avoids convoy effects between transactions (Blasgen et al. 1979) and ensure scalability under parallel workload. However, GPR data stores must sacrifice some form of consistency or provide little progress guarantees (Bushkov et al. 2014; Saeida Ardekani et al. 2013b; Attiya et al. 2009).

A data store API defines the shared data structures the client application processes manipulate as well as their consistency and progress guarantees. The above impossibility results inform the application developer that some APIs require synchronization among the data replicas. Process synchronization is costly; thus there is a trade-off between performance and data consistency.

Trade-Offs

In the common case, executing an operation under strong consistency requires to solve consensus among the data replicas, which costs at least one round trip among replicas (Lamport 2006). Sequential consistency allows to execute either read or write operations at a local replica (Attiya and Welch 1994; Wang et al. 2014). Weaker consistency models, e.g., eventual (Fekete et al. 1999) and strong eventual consistency (Shapiro et al. 2011), enable both read and write operations to be local.

A second category of trade-offs relates consistency models to metadata (Peluso et al. 2015; Burckhardt et al. 2014). They establish lower bounds on the space complexity to meet a certain consistency models. For instance, tracking causality accurately requires $O(m)$ bits of storage, where m is the number of replicas (Charron-Bost 1991).

Common Models

The previous sections introduce several consistency models (namely, SER, SC, LIN, SSER, and

EC). This section offers a perspective on other prominent models. Table 1 recapitulates.

D

Read Committed (RC)

Almost all existing transactional data stores ensure that clients observe only committed data (Zemke 2012; Berenson et al. 1995). More precisely, the RC consistency model enforces that if some read r observes the state \hat{x} of an item x in history h , then the transaction T_i that wrote \hat{x} commits in h . One can distinguish a *loose* and a *strict* interpretation of RC. The strict interpretation requires that $r(x)$ takes place after transaction T_i commits. Under the loose interpretation, the write operation might occur concurrently.

When RC, or a stricter consistency model holds, it is convenient to introduce the notion of *version*. A version is the state of a data item as produced by an update transaction. For instance, when T_i writes to some register x , an operation denoted hereafter $w(x_i)$, it creates a new version x_i of x . Versions allow to uniquely identify the state of the item as observed by a read operation, e.g., $r(x_i)$.

Strong Eventual Consistency (SEC)

Eventual consistency (EC) states that, for every data item x in the store, if there is no new update on x , eventually clients observe x in the same state. Strong eventual consistency (SEC) further constrains the behavior of the data replicas. In detail, a data store is SEC when it is EC, and moreover, for every item x , any two replicas of x that applied the same set of updates on item x are in the same state.

Client Monotonic (CM)

Client monotonic (CM) ensures that a client always observes the results of its own past operations (Terry et al. 1994). CM enforces the following four so-called session guarantees:

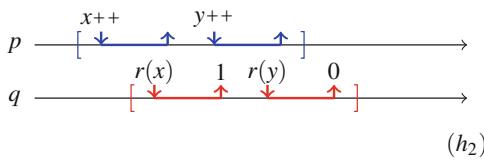
- (i) Monotonic reads (MR): if a client executes $r(x_i)$, then $r(x_{j \neq i})$ in history h , necessarily x_j follows x_i for some version order $\ll_{h,x}$ over the updates applied to x in h .

- (ii) Monotonic writes (MW): if a client executes $w(x_i)$, then $w(x_j)$, the version order $x_i \ll_{h,x} x_j$ holds;
- (iii) Read-my-writes (RMW): when a client executes $w(x_i)$ followed by $r(x_j \neq i)$, then $x_i \ll_{h,x} x_j$ holds;
- (iv) Writes-follow-reads (WFR): if a client executes $r(x_i)$ followed by $w(x_j)$ it is true that $x_i \ll_{h,x} x_j$.

Most consistency models require CM, but this guarantee is so obvious that it might be sometimes omitted – this is, for instance, the case in Gray and Reuter (1992).

Read Atomic (RA)

Under RA, a transaction sees either all of the updates made by another transaction or none of them (the All-or-Nothing guarantee). For instance, if a transaction T sees the version x_i written by T_i and transaction T_i also updates y , then T should observe at least version y_i . If history h fails to satisfy RA, a transaction in h exhibits a *fractured read* (Bailis et al. 2014). For instance, this is the case of the transaction executed by process q in history h_2 below.

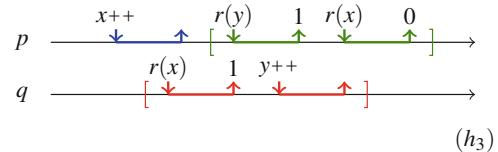


Consistent Snapshot (CS)

A transaction T_i depends on a transaction T_j when it reads a version written by T_j , or such a relation holds transitively. In other words, denoting $T_i \xrightarrow{\text{wr}} T_j$ when T_i reads from T_j , T_j is in the transitive closure of the relation $(\xrightarrow{\text{wr}})$ when starting from T_i .

When a transaction never misses the effects of some transaction it depends on, the transaction observes a *consistent snapshot* (Chan and Gray 1985). In more formal terms, a transaction T_i in a history h observes a consistent snapshot when for every object x , if (i) T_i reads version x_j , (ii) T_k writes version x_k , and (iii) T_i depends on

T_k , then version x_k is followed by version x_j in the version order $\ll_{h,x}$. A history h belongs to CS when all its transactions observe a consistent snapshot. For instance, this is not the case of history h_3 below. In this history, transaction $T_3 = r(y).r(x)$ depends on $T_2 = r(x).(y++)$, and T_2 depends on $T_1 = (y++)$, yet T_3 does not observe the effect of T_1 .



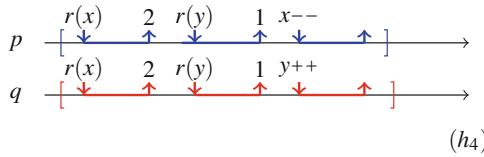
Causal Consistency (CC)

Causal consistency (CC) holds when transactions observe consistent snapshots of the system, and the client application processes are monotonic. CC is a weak consistency model, and it does not allow solving consensus. It is in fact the strongest model that is available under partition (Attiya et al. 2017). Historically, CC refers to the consistency of single operations on a shared memory (Ahamad et al. 1995). Causally consistent transactions (Causal HAT) is a consistency model that extends CC to transactional data stores (Bailis et al. 2013).

Snapshot Isolation (SI)

SI is a widely used consistency model (Berenson et al. 1995). This model is strong but allows more interleavings of concurrent read transactions than SER. Furthermore, SI is causal (i.e., $SI \subseteq CC$), whereas SER is not.

Under SI, a transaction observes a *snapshot* of the state of the data store at some point prior in time. Strong snapshot isolation (SSI) requires this snapshot to contain all the preceding transactions in real time (Daudjee and Salem 2006). Two transactions may commit under SI as long as they do not write the same item concurrent. SI avoids the anomalies listed in section “[Consistency Models](#)” but exhibits the *write-skew* anomaly, illustrated in history h_4 below.

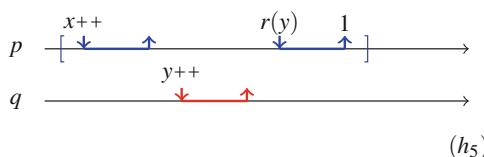


In this history, an application using data items x and y wishes to maintain the invariant $x \geq y$. The invariant holds initially and each of the two transactions T_1 and T_2 guarantees the invariant individually. As illustrated in history h_4 , running them concurrently under SI may violate the invariant.

An application is *robust* against a consistency model M when it produces serializable histories (Cerone and Gotsman 2016), despite running atop a data store providing M . It is known (Fekete et al. 2005) that an application is robust against SI when every invariant is materialized by a data item.

Parallel / Non-monotonic Snapshot Isolation (PSI/NMSI)

Parallel and non-monotonic snapshot isolation are scalable variations of SI. These models retain two core properties of SI, namely, (i) each transaction observes a consistent snapshot and (ii) no two concurrent transactions update the same data items. PSI requires to take a snapshot at the start of the transaction. NMSI relaxes this requirement, enabling the snapshot to be computed incrementally, as illustrated in history h_5 below.



A Three-Dimensional View of Data Consistency

Shapiro et al. (2016) classify consistency models along three dimensions, to better understand and compare them. Their approach divides each operation into two parts: the *generator* reads data and computes response values, and the *effector* applies side-effects to every replica. Each of

the three dimensions imposes constraints on the generators and effectors. Table 2 classifies the consistency criteria of Table 1 along these three dimensions.

D

- *Visibility dimension.* This dimension constrains the visibility of operations, i.e., how a generator sees the updates made by effectors. The strongest class of consistency models along this dimension is *external* visibility, which imposes that a generator sees the effectors of all the operations that precedes it in real time. Weakening this guarantee to the per-process order leads to *causal* visibility. A yet weaker class is *transitive* visibility, which only requires visibility to hold transitively. Finally, absence of constraints on generators, for instance, during the unstable period of an eventually consistent data store, is termed *rollback* visibility.
- *Ordering dimension.* This dimension constrains the ordering over generators and effectors. Four classes are of interest along this dimension: The strongest class is termed *total* order. For every history of a model in this class, there exists an equivalent serial history of all the operations. Weaker classes, below total order, constrain only effectors. The *gapless* order class requires effectors to be ordered online by natural numbers with no gaps; this requires consensus and is subject to the CAP impossibility result. The *capricious* class admits gaps in the ordering, allowing replicas to order their operations independently. A last-writer-wins protocol (e.g., Ladin et al. 1990) produces a consistency model in this class. This class is subject to the lost update anomaly. The weakest class along this dimension is termed *concurrent* and imposes no ordering on generators and effectors.
- *Composition dimension.* This dimension captures the fact that a transaction contains one or more operations. A model in the All-or-Nothing class preserves the A in ACID. This means that if some effector of transaction T_1 is visible to transaction T_2 , then all of T_1 's

Database Consistency Models, Table 2

Three-dimensional features of consistency models and systems

Acronym	Ordering	Visibility	Composition
EC	Capricious	Rollbacks	Single Operation
CM	Concurrent	Monotonic	Single Operation
CS	Concurrent	Transitive	All-or-Nothing + Snapshot
CC	Concurrent	Causal	Single Operation
Causal HAT	Concurrent	Causal	All-or-Nothing + Snapshot
LIN	Total	External	Single Operation
NMSI	Gapless	Transitive	All-or-Nothing + Snapshot
PSI	Gapless	Causal	All-or-Nothing + Snapshot
RC	Concurrent	Monotonic	All-or-Nothing
SC	Total	Causal	Single Operation
SER	Total	Transitive	All-or-Nothing + Snapshot
SI	Gapless	Transitive	All-or-Nothing + Snapshot
SSER	Total	External	All-or-Nothing + Snapshot
SSI	Gapless	External	All-or-Nothing + Snapshot

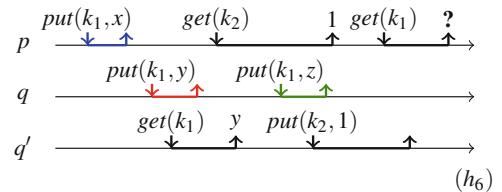
effectors are visible to T_2 . Typically, all the generators of a transaction read from the same set of effectors, i.e., its snapshot. The snapshot class extends the Visibility and Ordering guarantees to all generators of the transaction. For instance, in the case of a model both in the snapshot and total order classes, all the operations of a transaction are adjacent in the equivalent serial history.

Examples of Application

A key-value store (KVS) is a distributed data store that serves as building block of many cloud applications. This type of system belongs to the larger family of NoSQL databases and is used to store uninterpreted blobs of data (e.g., marshaled objects).

A KVS implements a map that is a mutable relation from a set of *keys* to a set of *values*. In detail, the API of a key-value store consists of two operations: Operation $put(k, v)$ adds the pair (k, v) to the mapping, updating it if necessary. Depending on the KVS, this operation may return the previous value of the key k , or simply *nil*. Operation $get(k)$ retrieves the current value stored under key k .

The notions of “current” and “previous” values depend on the consistency model of the KVS. History h_6 below illustrates this point for an operation $get(k_1)$ by process p .



When the KVS is RC, operation $get(k_1)$ returns the initial value of k_1 , or any value written concurrently or before this call. Denoting \perp the initial value of k_1 , this means that operation $get(k_1)$ may return any value in $\{\perp, x, y, z\}$.

If the KVS guarantees RMW, at least the last value written by p should be returned. As a consequence, the set of possible values reduces to $\{x, y, z\}$.

Now, let us consider that the KVS guarantees CC. Process p observes the operation $put(k_2, 1)$ by r . This operation causally follows an observation by q' of y . Therefore, p should observe either y or z .

If the KVS is linearizable, the value stored under key k_1 is the last value written before $get(k_1)$ in any sequential history equivalent to h_6 . Every such history should preserve the real-time precedence of h_6 . Clearly, the last update in h_6 sets the value of k_1 to z . Thus, if the KVS is linearizable, z is the only allowed response of operation $get(k_1)$ in h_6 .

Future Directions for Research

Consistency models are formulated in various frameworks and using different underlying assumptions. For instance, some works (ANSI 1992; Berenson et al. 1995) define a model in terms of the anomalies it forbids. Others rely on specific graphs to characterize a model (Adya 1999) or predicates over histories (Viotti and Vukolić 2016). The existence of a global time (Papadimitriou 1979) is sometimes taken for granted. This contrasts with approaches (Lamport 1986) that avoid to make such an assumption. A similar observation holds for concurrent operations which may (Guerraoui and Kapalka 2008) or not (Ozsu and Valduriez 1991) overlap in time.

This rich literature makes difficult an apples-to-apples comparison between consistency models. Works exist (Chrysanthis and Ramamritham 1994) that attempt to bridge this gap by expressing them in a common framework. However, not all the literature is covered, and it is questionable whether their definitions are equivalent to the ones given in the original publications.

The general problem of the implementability of a given model is also an interesting avenue for research. One may address this question in terms of the minimum synchrony assumptions to support a particular model. In distributed systems, this approach has led to the rich literature on failure detectors (Freiling et al. 2011). A related question is to establish lower and upper bounds on the time and space complexity of an implementation (when it is achievable). As pointed out in section “Trade-Offs”, some results already exist, yet the picture is incomplete.

From an application point of view, three questions are of particular interest. First, the robustness of an application against a particular consistency model (Fekete et al. 2005; Cerone and Gotsman 2016). Second, the relation between a model and a consistency control protocol. These two questions are related to the grand challenge of synthesizing concurrency control from the application specification (Gotsman et al. 2016). A third challenge is to compare consistency models in practice (Kemme and Alonso 1998; Wiesmann

and Schiper 2005; Saeida Ardekani et al. 2014), so as to understand their pros and cons.

D

Cross-References

- ▶ [Achieving Low Latency Transactions for Geo-Replicated Storage with Blotter](#)
- ▶ [Conflict-Free Replicated Data Types CRDTs](#)
- ▶ [Coordination Avoidance](#)
- ▶ [Data Replication and Encoding](#)
- ▶ [Databases as a Service](#)
- ▶ [Geo-Replication Models](#)
- ▶ [Geo-Scale Transaction Processing](#)
- ▶ [In-Memory Transactions](#)
- ▶ [Storage Hierarchies for Big Data](#)
- ▶ [TARDiS: A Branch-and-Merge Approach to Weak Consistency](#)
- ▶ [Weaker Consistency Models/Eventual Consistency](#)

References

- Adya A (1999) Weak consistency: a generalized theory and optimistic implementations for distributed transactions. Ph.d., MIT
- Ahamad M, Neiger G, Burns JE, Kohli P, Hutto PW (1995) Causal memory: definitions, implementation, and programming. *Distrib Comput* 9(1):37–49
- ANSI (1992) American national standard for information systems – database language – SQL. American National Standards Institute, New York
- Attiya H, Welch JL (1994) Sequential consistency versus linearizability. *ACM Trans Comput Syst* 12(2):91–122
- Attiya H, Bar-Noy A, Dolev D (1990) Sharing memory robustly in message-passing systems. Technical report MIT/LCS/TM-423, Massachusetts institute of technology, Laboratory for computer science, Cambridge
- Attiya H, Hillel E, Milani A (2009) Inherent limitations on disjoint-access parallel implementations of transactional memory. In: Proceedings of the twenty-first annual symposium on parallelism in algorithms and architectures, SPAA’09. ACM, New York, pp 69–78
- Attiya H, Ellen F, Morrison A (2017) Limitations of highly-available eventually-consistent data stores. *IEEE Trans Parallel Distrib Syst (TPDS)* 28(1): 141–155
- Bailis P, Davidson A, Fekete A, Ghodsi A, Hellerstein JM, Stoica I (2013) Highly available transactions: virtues and limitations. *Proc VLDB Endow* 7(3):181–192
- Bailis P, Fekete A, Hellerstein JM, Ghodsi A, Stoica I (2014) Scalable atomic visibility with ramp transac-

- tions. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD'14. ACM, New York, pp 27–38
- Berenson H, Bernstein P, Gray J, Melton J, O’Neil E, O’Neil P (1995) A critique of ANSI SQL isolation levels. *SIGMOD Rec* 24(2):1–10
- Blasgen M, Gray J, Mitoma M, Price T (1979) The convoy phenomenon. *ACM SIGOPS Oper Syst Rev* 13(2): 20–25
- Burckhardt S, Gotsman A, Yang H, Zawirski M (2014) Replicated data types: specification, verification, optimality. In: The 41st annual ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL’14, San Diego, 20–21 Jan 2014, pp 271–284
- Bushkov V, Dziuma D, Fatourov P, Guerraoui R (2014) The PCL theorem: transactions cannot be parallel, consistent and live. In: Proceedings of the 26th ACM symposium on parallelism in algorithms and architectures, SPAA’14. ACM, New York, pp 178–187
- Cerone A, Gotsman A (2016) Analysing snapshot isolation. In: Proceedings of the 2016 ACM symposium on principles of distributed computing, PODC, Chicago, 25–28 July 2016, pp 55–64
- Chan A, Gray R (1985) Implementing distributed read-only transactions. *IEEE Trans Softw Eng* SE-11(2):205–212
- Chandra TD, Hadzilacos V, Toueg S (1996) The weakest failure detector for solving consensus. *J ACM* 43(4):685–722
- Charron-Bost B (1991) Concerning the size of logical clocks in distributed systems. *Inf Process Lett* 39(1):11–16
- Chrysanthis PK, Ramamritham K (1994) Synthesis of extended transaction models using ACTA. *ACM Trans Database Syst* 19(3):450–491
- Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman J, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D (2012) Spanner: Google’s globally-distributed database. In: Symposium on operating system design and implementation (OSDI). Usenix, Hollywood, pp 251–264
- Daudjee K, Salem K (2006) Lazy database replication with snapshot isolation. In: Proceedings of the 32nd international conference on very large data bases, VLDB Endowment, VLDB’06, pp 715–726
- Delporte-Gallet C, Fauconnier H, Guerraoui R, Hadzilacos V, Kouznetsov P, Toueg S (2004) The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proceedings of the twenty-third annual ACM symposium on principles of distributed computing, PODC’04. ACM, New York, pp 338–346
- Fekete A, Gupta D, Luchangco V, Lynch N, Shvartsman A (1999) Eventually-serializable data services. *Theor Comput Sci* 220:113–156. Special issue on Distributed Algorithms
- Fekete A, Liarokapis D, O’Neil E, O’Neil P, Shasha D (2005) Making snapshot isolation serializable. *Trans Database Syst* 30(2):492–528
- Fischer MJ, Lynch NA, Patterson MS (1985) Impossibility of distributed consensus with one faulty process. *J ACM* 32(2):374–382
- Freiling FC, Guerraoui R, Kuznetsov P (2011) The failure detector abstraction. *ACM Comput Surv* 43(2):9:1–9:40
- Gilbert S, Lynch N (2002) Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33(2): 51–59
- Gotsman A, Yang H, Ferreira C, Najafzadeh M, Shapiro M (2016) ‘Cause I’m strong enough: reasoning about consistency choices in distributed systems. In: Proceedings of the 43rd annual ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL’16. ACM, New York, pp 371–384
- Gray J, Reuter A (1992) Transaction processing: concepts and techniques, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco
- Gray J, Reuter A (1993) Transaction processing: concepts and techniques. Morgan Kaufmann, San Francisco, ISBN:1-55860-190-2
- Guerraoui R, Kapalka M (2008) On the correctness of transactional memory. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP’08. ACM, New York, pp 175–184
- Herlihy M, Wing J (1990) Linearizability: a correctness condition for concurrent objects. *ACM Trans Program Lang Syst* 12(3):463–492
- Israeli A, Rappoport L (1994) Disjoint-access-parallel implementations of strong shared memory primitives. In: Proceedings of the thirteenth annual ACM symposium on principles of distributed computing, PODC’94. ACM, New York, pp 151–160
- Kemme B, Alonso G (1998) A suite of database replication protocols based on group communication primitives. In: International conference on distributed computing systems, IEEE. IEEE computer society, pp 156–163
- Ladin R, Liskov B, Shrira L (1990) Lazy replication: exploiting the semantics of distributed services. In: IEEE computer society technical committee on operating systems and application environments, IEEE, vol 4. IEEE computer society, pp 4–7
- Lamport L (1979) How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans Comput* 28(9):690–691
- Lamport L (1986) On interprocess communication. Part I: basic formalism. *Distrib Comput* 1(2): 77–85
- Lamport L (2006) Lower bounds for asynchronous consensus. *Distrib Comput* 19(2): 104–125
- Murty J (2008) Programming Amazon web services, 1st edn. O’Reilly, Sebastopol

- Ozsu MT, Valduriez P (1991) Principles of distributed database systems. Prentice-Hall, Inc., Upper Saddle River
- Papadimitriou CH (1979) The serializability of concurrent database updates. *J ACM* 26(4):631–653
- Peluso S, Palmieri R, Romano P, Ravindran B, Quaglia F (2015) Disjoint-access parallelism: impossibility, possibility, and cost of transactional memory implementations. In: Proceedings of the 2015 ACM symposium on principles of distributed computing, PODC’15. ACM, New York, pp 217–226
- Saeida Ardekani M, Sutra P, Shapiro M (2013a) Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems. In: Symposium on reliable distribution system (SRDS). IEEE computer society, Braga, pp 163–172
- Saeida Ardekani M, Sutra P, Shapiro M (2013b) On the scalability of snapshot isolation. In: Proceedings of the 19th international euro-par conference (EUROPAR)
- Saeida Ardekani M, Sutra P, Shapiro M (2014) G-DUR: a middleware for assembling, analyzing, and improving transactional protocols. In: Proceedings of the 15th international middleware conference, Middleware’14. ACM, New York, pp 13–24
- Schiper N, Sutra P, Pedone F (2010) P-store: genuine partial replication in wide area networks. In: Proceedings of the 29th IEEE international symposium on reliable distributed systems (SRDS)
- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) Conflict-free replicated data types. In: Défago X, Petit F, Villain V (eds) International symposium on stabilization, safety, and security of distributed system (SSS). Lecture notes in computer science, vol 6976. Springer, Grenoble, pp 386–400
- Shapiro M, Saeida Ardekani M, Petri G (2016) Consistency in 3D. In: Desharnais J, Jagadeesan R (eds) International conference on concurrency theory (CONCUR), Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Dagstuhl Publishing, Leibniz international proceeding in informatics (LIPICS), Québec, vol 59, pp 3:1–3:14
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Symposium on operating system principles (SOSP). Association for Computing Machinery, Cascais, pp 385–400
- Terry DB, Demers AJ, Petersen K, Spreitzer MJ, Theimer MM, Welch BB (1994) Session guarantees for weakly consistent replicated data. In: International conference on parallel and distributed information system (PDIS), Austin, pp 140–149
- Viotti P, Vukolić M (2016) Consistency in non-transactional distributed storage systems. *ACM Comput Surv* 49(1):19:1–19:34
- Vogels W (2008) Eventually consistent. *ACM Queue* 6(6):14–19
- Wang J, Talmage E, Lee H, Welch JL (2014) Improved time bounds for linearizable implementations of abstract data types. In: 2014 IEEE 28th international parallel and distributed processing symposium, pp 691–701
- Wiesmann M, Schiper A (2005) Comparison of database replication techniques based on total order broadcast. *IEEE Trans Knowl Data Eng* 17(4):551–566
- Zemke F (2012) What’s new in SQL:2011. *SIGMOD Rec* 41(1):67–73

D

Databases as a Service

Renato Luiz de Freitas Cunha
IBM Research, São Paulo, Brazil

Synonyms

[Cloud Databases](#)

Definitions

Cloud provider: A cloud infrastructure provider, such as Amazon Web Services, Microsoft Azure, or IBM Cloud.

End user: A user of cloud services, a person or company that uses services from the cloud provider.

Database server: A database management system that uses the client-server model.

Database as a service: Cloud computing service model that abstracts the setup of hardware, software, or tuning for providing access to a database.

Acronyms

IAAS Infrastructure as a service

PAAS Platform as a service

SAAS Software as a service

DBAAS Database as a service

IT Information technology

DBA Database administrator

API Application programming interface

Overview

Configuring and maintaining database systems is a complicated endeavor, which includes planning the hardware that will be used, configuring disks and file systems, installing and tuning

database management systems, determining whether and how replication will be done, and managing backups. In traditional Information technology (**IT**) environments, the setup of database systems requires the work of different professionals, such as Database administrators (**DBAs**), systems administrators, and storage administrators. In fact, tuning databases requires broad knowledge and a principled approach for measuring performance (Shasha and Bonnet 2002), and a well-tuned database system can perform faster than an improperly tuned system by a factor of two or more (Stonebraker and Cattell 2011).

In a Database as a service (**DBaaS**) setting, a service provider hosts databases providing mechanisms to create, store, and access databases at the provider. The means of interacting with the service are usually via web interfaces and Application programming interfaces (**APIs**). Although popularized in recent years by cloud computing and big data, this is not a new concept and, in fact, has been available for more than a decade (Hacigumus et al. 2002).

More recently, cloud providers have made **DBaaS** offerings available. In a **DBaaS** model, application developers neither have to maintain machines nor have to install and maintain database management systems themselves. Instead, the database service provider is responsible for installing and maintaining database systems, and users are charged according to their usage of the service.

To better place the **DBaaS** model in the cloud computing service models, a quick review of the cloud computing definition will be presented. Interested readers are directed to other sources (Mell and Grance 2011; Armbrust et al. 2010) for a more detailed definition. Readers familiar with cloud computing concepts can safely skip the next section.

Defining Characteristics of Cloud Computing

The essential characteristics of the cloud computing model is that it has *broad network access*, with *resource pooling*, *rapid elasticity*, and *measured services*, and provides *on-demand*

self-service (Mell and Grance 2011). Broad network access is a necessity in clouds, for cloud providers can be geographically distant from end users, and even in this case, end users rely on cloud services. In clouds, resources are pooled, meaning that resources are already available in the provider and, upon request, are assigned to users. These resource pools enable the rapid elasticity in clouds, enabling the addition and removal of resources in a short amount of time.

Services in clouds can be provided in three main models:

*Software as a service (**SaaS**)* In this model, applications execute on a cloud infrastructure, and users normally access them from a web browser. Usually, consumers are end users, who will not do development on top of such applications.

*Platform as a service (**Paas**)* In this model, application developers and service providers are able to deploy onto the cloud infrastructure applications created using programming languages, libraries, tools, and services supported by the cloud provider.

*Infrastructure as a service (**IaaS**)* In this model, customers have access to computing, storage, network, and similar services where the customer can execute arbitrary operating systems and software.

In this entry, there is no distinction between private and public clouds, for the discussion will be made considering the perspective of the end user and the cloud provider. In private clouds, the infrastructure is provisioned for exclusive use by a single organization. In this model, different business units are usually the end users of cloud services. In public clouds, the infrastructure is provisioned for open use by the general public.

Positioning **DBaaS**

IaaS allows one to abstract the actual hardware configuration away, but the issues for properly configuring databases remain. Hence, end users still have to devote time and energy on properly configuring and maintaining database systems, when such energy would be better applied in

developing the system they wanted to write in the first place.

From the cloud computing definition, **DBaaS** has aspects of both **SaaS** and **Paas**: although databases are software packages, they expose a programming interface, the query language, with which developers interact with. In public clouds, **DBaaS** usually is offered in the **Paas** layer.

Advantages of the DBaaS Model

There are many advantages in adopting a **DBaaS** model, all of them resulting from using a cloud-based service model. In **DBaaS**, due to the self-service nature of the cloud, to create new database instances, a user only needs to make a request in the cloud administration portal. Also, there is no need to plan for machines, storage, or replication strategies, since cloud providers either provide options to enable replication automatically in different zones or manage storage transparently, depending on provider and service.

The agility of cloud deployments should be contrasted with traditional **IT** operations. Consider, for example, that a business unit is developing a new application that requires a database server connection. To set up the server, a system administrator would have to be involved to allocate the machine and install the database server for the business unit. Between requesting a database server and actually being able to use it, the business unit could have to wait one or two business days. In the cloud, due to its high levels of automation, a new database instance can be provisioned in a matter of minutes.

Assuming the business unit of the previous example used **IaaS** as a model for installing database servers, whoever provisioned the computing resource would still be required to implement corporate policies, such as creating different accounts for different users and configuring passwords and firewalls according to corporate policy. Essentially, the business unit would still need access to a system administrator to properly configure the computing resource. Some of the work involved in this can be reduced by using

virtual machine templates, but, given the control users can have over virtual machines, some human verification may still be required.

What makes **DBaaS** stand out is the heavy use of abstraction and automation, another characteristic of the cloud. To provision a new database server instance, the user specifies the database engine and version required and the amount of RAM and the amount of storage required, and the cloud provider takes care of actually provisioning the database instance.

In principle, **DBaaS** works well for both small and large users. Teams that need small databases benefit from the fast provisioning times and from the lack of requiring a **DBA**, while teams with large database requirements benefit from the ability to scale up instances whenever needed.

Disadvantages of the DBaaS Model

Although **DBaaS** is a good solution for many companies and teams, it is not without its issues. Users might need database extensions not available in the cloud provider. Consider, for example, PostGIS. PostGIS is an extension that adds support for geographic objects to the PostgreSQL database. To add this support, PostGIS is installed as a set of helper programs, shared library files, and SQL scripts. If an end user needs PostGIS enabled in a database, but the cloud provider does not support it, then the user has no means of adding geographic object support for that database instance. The same argument can be made about legacy versions of database servers or unpopular database engines: if the cloud provider does not support them, they cannot be used.

With regard to scaling, not all providers might support scaling down database server instances. If, for example, a user provisioned a database instance larger than what was actually needed, the only solution to scaling down might be provisioning a new database instance and migrating the data from the old instance to the new one. Scaling database services out might suffer from a similar issue. If the cloud provider lacks support

for multiple server instances serving the same database – this can occur, for example, when a database grows to be bigger than the biggest instance type supported by the cloud provider – then the user has to implement support manually, i.e., by partitioning the data among different database servers.

Depending on the needs of the end user, the premium for using **DBaaS** might be higher than deploying databases traditionally in **IaaS** clouds or on in-house machines. Therefore, a cost-benefit analysis must be made prior to choosing one option or another.

Types of Databases

Currently there are two dominant database paradigms: relational (SQL) and the so-called NoSQL databases. Relational databases are based on the relational calculus theory and make use of the structured query language (SQL). Different relational databases implement different dialects of the SQL, but the language itself is standardized, and SQL databases make use of similar terminology. Relational databases provide ACID (atomicity, consistency, isolation, and durability) guarantees:

Atomicity requires that each transaction happen all at once or not at all – if one part of the transaction fails, then the whole transaction fails – to the outside world, committed transactions appear indivisible;

Consistency ensures that transactions change the database only between valid states, making sure that all defined rules, such as cascades, constraints, and triggers, are always valid;

Isolation ensures that one transaction cannot interfere with another and that the concurrent execution of transactions results in the same state that would be obtained if they were executed sequentially;

Durability ensures that once a transaction has been committed, it will remain committed, even in the event of crashes.

The term NoSQL is newer and relates to databases with different data models, such as

tuples (where row fields are predefined in a schema), documents (which allow attributes to be nested documents or lists of values as well as scalars and without the definition of a schema), extensible records (hybrids between tuples and documents), and objects (analogous to objects in programming languages, but without methods) (Cattell 2011). Opposed to relational databases, NoSQL databases provide BASE (basically available, soft state, eventually consistent) properties.

Basically available meaning that the system guarantees availability, in which every request receives a response, without guarantee of containing the most recent write

Soft state means that state can change, even without input, due to eventual consistency;

Eventually consistent indicates the system will become consistent over time, given it does not receive input during that time.

NoSQL databases became popular due to big Internet companies, such as Amazon, Facebook, and Google, using them for scaling their services. Traditionally, relational databases were considered hard to scale out, particularly due to the ACID guarantees. Hence, companies with millions of users needed better ways to scale their systems. In a way, these database systems were introduced to address the “velocity” and “volume” of the big data Vs. Examples of early successful systems are Dynamo (DeCandia et al. 2007) and BigTable (Chang et al. 2008).

The gains in performance of NoSQL databases over SQL databases come from the BASE properties. By relaxing the ACID properties, NoSQL databases can return from queries faster, allowing for faster responses. Also, the eventual consistency allows for easier load balancing between back-end servers, since the global view does not have to be consistent the whole time. In fact, “customers tolerate airline over-booking, and orders that are rejected when items in an online shopping cart are sold out before the order is finalized” (Cattell 2011), which makes a case for using such databases.

Databases as a Service, Table 1 Non-exhaustive list of **DBaaS** offerings of three big cloud providers as of the fourth quarter of 2017. Table sorted by provider name

Provider	Type	Databases supported
Amazon	Relational	Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, Microsoft SQL Server
Amazon	NoSQL	Amazon SimpleDB, Amazon Redshift, Amazon DynamoDB, Amazon Neptune, JanusGraph
Google	Relational	PostgreSQL, MySQL, Cloud Spanner
Google	NoSQL	Cloud Bigtable, Cloud Datastore
IBM	Relational	IBM DB2, PostgreSQL, MySQL
IBM	NoSQL	IBM Cloudant, MongoDB, JanusGraph, Redis, RethinkDB, ScyllaDB

D

Current DBaaS Offerings

As of the fourth quarter of 2017, all the big cloud providers have a **DBaaS** offering in their cloud systems, whether relational or NoSQL. Table 1 displays a non-exhaustive list of the offerings of three big cloud providers. As can be seen in the table, various databases with different data models are available, allowing users to select the model that best fits the problem being tackled.

Relational database management systems achieve scalability by partitioning data between different database servers, a technique known as “sharding.” One example of how sharding can be done is by splitting the data based on the continent users live in – e.g., the data that belongs to a user from Europe would be placed on a server different from a server that stores data about a user from Asia. When a user migrates between continents, migrating user data between regions would reduce access times but would incur data transfer costs. One interesting study would be to analyze the trade-off between higher latencies for users and the costs of such migrations. Also related to costs, cloud providers could benefit from more accurate cost models that better reflect their fail-over strategies (Mansouri 2017).

In recent years, we have seen database management systems originally written as NoSQL databases adopting SQL as a query language and becoming relational databases (Corbett et al. 2013; Bacon et al. 2017). We have also seen new relational databases being written specifically to work on the cloud (Verbitski et al. 2017). Such systems were designed to overcome the technical limitations of both traditional relational (which were hard to scale horizontally due to the inherent complexity of sharding) and NoSQL databases (which typically only provide eventual consistency), while providing ACID guarantees and without the need of performing explicit sharding. These new relational databases were made generally available to public cloud customers, but currently there seem to be no private cloud alternatives to such database systems. It would

DBaaS and Big Data

Advances in database technologies enabled collection and analysis for big data. As the ideas used by Internet companies to implement scalable database services became popular, application developers started using such technologies to implement faster applications. Today, **DBaaS** enables companies from various industries to collect, process, and analyze big data in the cloud.

Future Directions for Research

Research in **DBaaS** is related to the areas discussed previously: relational and NoSQL databases and cloud computing. Therefore, it makes sense that future research directions should be related to a tighter integration between these areas.

be interesting to investigate how cloud software stacks (e.g., OpenStack (Sefraoui et al. 2012)) could integrate these improvements and whether current **DBaaS** software architectures (as implemented by cloud providers) could accommodate these new designs.

As for the perceived “rigidity” of relational databases and SQL, vendors of database management systems have added support for a JSON data type to their products, resulting in more flexibility in the programming model, as such data types add support for the JSON syntax, allowing for more complex queries without the need for changing the database schema every time a new field is modified (in the JSON columns). Previous research (Chasseur et al. 2013; Tahara et al. 2014) suggests that such integration of SQL and NoSQL properties results in systems with higher performance and better consistency guarantees than only relational or only NoSQL systems, but may need further evaluation on Internet-scale and **DBaaS** systems.

Further Reading

Hacigumus et al. (2002) describe the design of one of the first web-based **DBaaS** systems before the term cloud computing was coined. Mell and Grance (2011) and Armbrust et al. (2010) define the term “cloud computing” and discuss the characteristics of the cloud. DeCandia et al. (2007) and Chang et al. (2008) describe two early successful NoSQL database systems. Cattell (2011) examines SQL and NoSQL systems and categorizes NoSQL according to data model while also presenting use cases for each of the defined categories. Abourezq and Idrissi (2016) present an overview of database as a service for big data, reviewing the database solutions offered as **DBaaS**.

Cross-References

- ▶ [Big Data in the Cloud](#)
- ▶ [Graph Processing Frameworks](#)
- ▶ [NoSQL Database Systems](#)

References

- Abourezq M, Idrissi A (2016) Database-as-a-service for big data: an overview. *Int J Adv Comput Sci Appl (IJACSA)* 7(1):157–177
- Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. *Commun ACM* 53(4):50–58. <https://doi.org/10.1145/1721654.1721672>, <http://doi.acm.org/10.1145/1721654.1721672>
- Bacon DF, Bales N, Bruno N, Cooper BF, Dickinson A, Fikes A, Fraser C, Gubarev A, Joshi M, Kogan E, et al (2017) Spanner: becoming a SQL system. In: Proceedings of the 2017 ACM international conference on management of data. ACM, pp 331–343
- Cattell R (2011) Scalable SQL and NoSQL data stores. *SIGMOD Rec* 39(4):12–27. <http://doi.org/10.1145/1978915.1978919>, <http://doi.acm.org/10.1145/1978915.1978919>
- Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. *ACM Trans Comput Syst (TOCS)* 26(2):4
- Chasseur C, Li Y, Patel JM (2013) Enabling JSON document stores in relational systems. In: WebDB, vol 13, pp 14–15
- Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P, et al (2013) Spanner: Google’s globally distributed database. *ACM Trans Comput Syst (TOCS)* 31(3):8
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon’s highly available key-value store. *ACM SIGOPS Oper Syst Rev* 41(6):205–220
- Hacigumus H, Iyer B, Mehrotra S (2002) Providing database as a service. In: Data engineering, 2002. Proceedings 18th international conference on IEEE, pp 29–38
- Mansouri Y (2017) Brokering algorithms for data replication and migration across cloud-based data stores. PhD thesis
- Mell P, Grance T (2011) The NIST definition of cloud computing. Technical report 800-145, Computer security division, Information technology laboratory, National institute of standards and technology gaithersburg
- Sefraoui O, Aissaoui M, Eleuldj M (2012) Openstack: toward an open-source solution for cloud computing. *Int J Comput Appl* 55(3):38–42
- Shasha D, Bonnet P (2002) Database tuning: principles, experiments, and troubleshooting techniques. Morgan Kaufmann, Amsterdam
- Stonebraker M, Cattell R (2011) 10 rules for scalable performance in “simple operation” datastores. *Commun ACM* 54(6):72–80

- Tahara D, Diamond T, Abadi DJ (2014) Sinew: a SQL system for multi-structured data. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 815–826
- Verbitski A, Gupta A, Saha D, Brahmadesam M, Gupta K, Mittal R, Krishnamurthy S, Maurice S, Kharatishvili T, Bao X (2017) Amazon aurora: design considerations for high throughput cloud-native relational databases. In: Proceedings of the 2017 ACM international conference on management of data. ACM, pp 1041–1052

Data-Driven Process Simulation

Benoît Depaire and Niels Martin
Research group Business Informatics, Hasselt University, Diepenbeek, Belgium

Definitions

Data-driven process simulation is a technique which constructs a computer model that imitates the internal details of a business process and extensively uses data – recorded by information systems supporting the actual process – to do so. The model is used to execute what-if scenarios in order to better understand the actual process behavior and predict the impact of potential changes to the process.

Overview

Data-Driven Process Simulation

Every organization executes multiple business processes – e.g., the production, transportation, and billing process – which have to be managed properly to generate customer value (Dumas et al. 2013). An essential part of business process management is the identification and design of process improvement opportunities – e.g., hire more staff to reduce waiting time at a specific step in the process. Since a business process typically has a complex and dynamic nature, it is often impossible to deduce analytically the full impact of a specific process change. In such setting, process simulation is a valuable instrument.

Process simulation is the imitation of the actual process through the use of a (computer) model. It allows an organization to verify the consequences of proposed process modifications prior to implementation (Melão and Pidd 2003). The quality of the results of process simulation is directly dependent on the quality of the process simulation model.

Therefore, to construct a high-quality simulation model, the researcher must gather extensive process information as input. Traditionally, such input is gathered by means of business documents, interviews, and observations. While valuable, it has been shown that these sources also have limitations:

- Process documentation often describes how a process should be executed which can deviate from the actual process behavior (Märüster and van Beest 2009).
- Interviews with business experts can result in contradictory information (Vincent 1998), and the perception of the interviewees might be heavily influenced by recent experiences within the process (Dickey and Pearson 2005).
- Observational data could suffer from the Hawthorne effect (Robinson 2004) – i.e., the performance of staff members increase simply because they know they are being observed – and often require a significant time investment.

Another type of input are the data and records kept by different information systems about the daily process execution. With the rise of process-aware information systems – supporting the entire process rather than a specific activity (Dumas et al. 2005), e.g., enterprise resource planning systems – the amount of these process data is growing rapidly. The benefit of these data is that they provide an objective view on the real process behavior. The challenge, however, is to extract insights from these data which can be used to construct more realistic simulation models.

The goal of data-driven process simulation is to extend the traditional inputs for simulation model construction with such process data and

to extract insights from the data with respect to different aspects of the simulation model. As a consequence, it encompasses several techniques which support specific modeling decisions.

Building Blocks of a Simulation Model

When constructing a simulation model, four building blocks can be distinguished: entities, activities, control-flow, and resources (Martin et al. 2016b). For each of these building blocks, a series of modeling tasks need to be executed during the construction of a simulation model.

An **entity** is the dynamic object that flows through the process model and which instigates action in the process as activities are executed on it by resources (Kelton et al. 2015; Tumay 1996) – e.g., a specific order (entity) in an order picking process. During simulation, multiple entities can be simultaneously present within the process and can influence each other's process flow by, e.g., occupying particular resources.

An **activity** is the execution of a specific task by a specific resource on a given entity (Tumay 1996) – e.g., the registration (task) by employee X (resource) of order Y (entity).

The **control-flow** of a process expresses the temporal relationship between the activities. Control-flow consists of sequence flows – modeling “directly follows” relationships – and gateways – modeling complex process behavior such as choice and parallelism. The control-flow defines the different routes an entity can follow through the process – e.g., an international order might follow a different route in the order picking process than a national order, resulting in different or additional activities that need to be executed on the order.

The actual execution of activities is the responsibility of **resources**. Basic resource types are technical resources, such as software systems, and human resources (Dumas et al. 2013; Tumay 1996).

Key Research Findings

This section provides the key research results on how process data can be used to improve different

modeling aspects of process simulation. It should be noted that, in general, the insights that can be retrieved from process data depend upon the quality and granularity of the available data. To structure the discussion in this section, the four building blocks of a simulation model and their associated modeling decisions are used.

Entities

With respect to the *entity* building block, two modeling decisions will be discussed: the arrival rate and the entity typology.

Entity Arrival Rate

Modeling the entity arrival rate – i.e., the pace at which new entities arrive at the process – typically consists of a deterministic and a stochastic part. The deterministic part can be a constant or dependent on other context-related factors. The stochastic element is typically modeled as a probability distribution (van der Aalst 2015).

Process data is used to learn the parameters of a predefined probability distribution. The exponential distribution is often used under the assumption that the start of the first activity in a process execution defines the actual arrival of the entity and the entities arrive independent from each other (Rozinat et al. 2009). When these assumptions appear too restrictive – e.g., an entity arrived before the first activity was executed on it – the ARPRA algorithm (Martin et al. 2016c) accounts for queue formation before the first activity and discovers the proper gamma distribution – which is a generalization of the exponential distribution.

Entity Types

No two entities are exactly the same, and it is not uncommon that entity properties have a direct impact on the process execution – e.g., the physical dimensions of an order have a direct impact on the duration of the packaging activity. To simplify complex processes, it is common to define entity types which represent a group of entities similar in terms of specific characteristics (i.e., attributes) (Kelton et al. 2015).

When working with entity types, the quality of the defined typology has a direct influ-

ence on the quality of the simulation model. To identify relevant entity types, cluster analysis is a useful technique to segment the entities (Xu and Wunsch 2005). Recently, various algorithms have been developed that discover clusters which group entities that have a common execution path and where clusters – entity types – differ in terms of process execution. Various approaches exist: a vector-based approach where the trajectory of an entity is translated into a set of features (Bose and van der Aalst 2010; de Medeiros et al. 2008; Delias et al. 2015; Greco et al. 2006; Song et al. 2009), an approach based on the generic edit distance (Bose and van der Aalst 2009), Markov model-based clustering (Veiga and Ferreira 2010), and an approach driven by process model accuracy (De Weerdt et al. 2013).

Activities

With respect to activities, five important modeling decisions will be discussed: determining which activities need to be included, activity duration, resource requirements, queue discipline, and activity interruptions.

Activity Definition

Process data is typically captured at the level of events – i.e., specific moments in time which change the state of the process. Even when data is logged at the level of activities, this data might be at a too granular level for process simulation purposes. In both cases, techniques are needed to identify appropriate activities and to map data to higher-level activities. For this purpose, activity mining (Günther et al. 2010) can be used and only requires the event log. Other approaches which incorporate domain knowledge are also available such as in Szymanski et al. (2013), Baier et al. (2014), and Mannhardt et al. (2016).

Activity Duration

Modeling the activity duration typically consists of a deterministic and stochastic element. Deterministic differences in activity duration can be dependent on a wide range of entity properties but also on resource workload (cf. Pospisil and Hruška (2012) and Nakatumba and van der Aalst (2010); Nakatumba et al. (2012)). Process data

can be used to discover these dependencies and model them correctly.

The stochastic element is modeled by means of a probability distribution. Activity durations are extracted from the process data – i.e., difference between the start and completion time of an activity – and used to fit the probability distribution.

When either start or completion time of activities is missing, this approach cannot be applied directly. Literature suggest several approaches to deal with such missing data. When only the completion time is recorded, various proxies are suggested for the corresponding start time, such as the completion time of the previous activity for that entity or the time at which the resource executing the activity finished its previous job (Nakatumba 2013; Wombacher and Iacob 2013). When only the start of an activity is recorded, similar proxies have been suggested in Martin et al. (2016b). Be aware that several issues such as the interruptibility of activities or the presence of batch processing can render the extraction of duration observations more complex.

Resource Requirements

The resource(s) required to execute a particular activity need(s) to be specified. Process data containing resource information can be used to support this modeling task, and different techniques exist to reveal the relationship between activities and resources (Huang et al. 2011; Liu et al. 2008; Ly et al. 2006; Senderovich et al. 2014).

Queue Discipline

In a simulation model, one must define how queues are dealt with at activities. The queue discipline establishes the processing order of queuing entities, such as first-in-first-out (FIFO), last-in-first-out (LIFO), or according to a priority rule.

Process data can be used to discover the queuing discipline that is applied. Suriadi et al. (2017) propose a method to extract FIFO, LIFO, and priority-based processing patterns from process-related data and consider three perspectives: the resource, activity, and case perspective, each representing potential levels of aggregation.

Activity Interruptibility and Unexpected Interruptions

With respect to interruptions, the activity interruptibility and unexpected interruptions need to be modeled. The activity's interruptibility expresses whether it can be interrupted while an entity is being processed. If activities are interruptible, one must still model the interruption frequency, duration, and nature. The nature of an interruption refers to whether the activity execution stops immediately (preemptive) or whether the running entity can still be finished (non-preemptive) (Hopp and Spearman 2011).

Process data containing specific information such as interruption events or resource break information (Senderovich et al. 2014) can be used directly to inform the aforementioned modeling tasks. However, often such specific information is missing, and the interruption time is simply part of the activity duration. Under these circumstances, duration outlier analysis can be used (Pika et al. 2013; Rogge-Solti and Kasneci 2014) to inform the researcher on activity interruptions.

Control-Flow

With respect to the control-flow, two aspects need to be modeled: the control-flow definition between the activities and the gateway routing logic.

Control-Flow Definition

The control-flow determines the path(s) an entity can follow throughout the process and defines sequentiality, choice, and concurrency between activities. A multitude of algorithms exist to retrieve control-flow models from process data (De Weerdt et al. 2012; van der Aalst 2016). Several of these algorithms have been applied in a simulation context such as the alpha-algorithm (Aguirre et al. 2013; Rozinat et al. 2008, 2009), fuzzy mining (van Beest and Mărușter 2007), and heuristic mining (Leyer and Moermann 2015; Mărușter and van Beest 2009).

Gateway Routing Logic

For gateways representing choice, the routing logic needs to be defined, which can be stochastic

(each outgoing path has a specific probability), deterministic (business rules govern the choice), or a combination of both.

In literature, Rozinat and van der Aalst (2006a,b) use decision trees to discover the decision logic from process data, which is also applied in a simulation context (Rozinat et al. 2008, 2009). Schonenberg et al. (2010) use regression analysis to mine decision logic from process data. The work of de Leoni et al. (2013) allows one to mine rules from process data which are linear combinations of multiple entity attributes. All these approaches result in deterministic rules but can be easily extended with a stochastic element as suggested by Pospisil and Hruška (2012). A purely stochastic approach is presented in Leyer and Moermann (2015) which simply learns the routing probabilities for each choice.

Resources

With respect to modeling the resources, three modeling tasks are discussed: the definition of resource roles, resource schedules, and batch processing.

Resource Roles

Resource roles group resources performing similar activities and allow one to simplify the simulation model – e.g., activities are assigned to roles instead of specific resources.

Various work on how to retrieve insights on resource roles from process data exist. One approach is to perform a cluster analysis on resource-activity frequency matrix retrieved from the process data (Song and van der Aalst 2008; Rozinat et al. 2009). Another approach is to focus on handover relationships in the data (Burattin et al. 2013) or to cluster resources on the number of entities they collaborated on (Ferreira and Alves 2012).

Resource Schedules

A resource schedule expresses its availability for the simulated process, which can differ from the official working schedule as a resource is often involved in multiple processes. Process data can

help to gather insights in the resource's availability for the process.

A first approach to use process data is to estimate the start and end time of a working day from the process data (Wombacher et al. 2011). Other approaches estimate resource availability as a percentage of the total time a resource is executing activities (Liu et al. 2012; van der Aalst et al. 2010), which however do not provide exact availability timeframes. If more fine-grained insights are needed, the work of Martin et al. (2016a) provides a method to mine daily availability records for resources.

Batch Processing

Batch processing refers to a resource's tendency to accumulate entities in order to process them simultaneously, concurrently, or sequentially. Again, process data can be used to study the actual batching behavior.

Wen et al. (2013) developed a control-flow discovery algorithm when simultaneous batch processing occurs in a process. Similar to Liu and Hu (2007), they assume that events are only recorded for one of the cases in a batch. While Wen et al. (2013) focus on the control-flow, Nakatumba (2013) states that batching behavior is present when two sets of activity instances on a particular working day are separated by a time period of more than 1 hour. A more systematic approach is proposed in Martin et al. (2017). Based on a distinction between simultaneous, concurrent, and sequential batch processing, an algorithm is proposed which detects batching behavior. Moreover, metrics such as the batch size are calculated which can be used to specify batching parameters in a simulation model (Martin et al. 2017).

Examples of Application

Literature provides examples in which real-life process data is extensively used to support the construction of simulation models. For example, Rozinat et al. (2009) consider the complaint handling process and the invoice handling process in municipalities and use process data to specify the entity arrival rate, activity duration,

control-flow, gateway routing logic, and resource roles. In another example, Leyer and Moormann (2015) simulate the loan application process at a financial institution and use real-life process data to model the activity duration, control-flow, and gateway routing logic.

While the aforementioned references use process data to support multiple modeling decisions, dedicated papers focusing on a single decision also apply their proposed techniques to real-life data. For instance, Mărușter and van Beest (2009) mainly focus on the use of process data to model the simulation model's control-flow and consider three real-life processes: a process of a gas company, the fine collection process of a governmental institution, and the navigation process in an agricultural decision support system. In another example, Martin et al. (2017) retrieve insights in resources' batching behavior from a call center's and a production department's process data.

D

Future Directions for Research

Despite the valuable research efforts outlined above, four key directions of additional work to exploit process data for the development of a simulation model can be distinguished.

Firstly, there is room for new, improved, and dedicated algorithms which support specific modeling decisions using process data. This article only discusses a subset of modeling tasks required to build a simulation model. For other modeling tasks, e.g., the additional ones included in Martin et al. (2016b), no clear starting points for future research could be identified in literature. For some modeling decisions included in this article, the state of the art refers to approaches which were not directly developed with simulation in mind. For instance, the use of clustering to identify entity types results in distinct process models for each entity type, requiring an additional aggregation step to produce a single model. Dedicated algorithms can exploit the specific characteristics of process data and specific needs of process simulation. Additionally, existing (dedicated) algorithms can also be extended to provide even more insights.

For example, the ARPRA algorithm (Martin et al. 2016c) can be extended to discover nonstationary arrival patterns.

Secondly, a common assumption among process discovery techniques is that both start and completion of an activity are logged. Often the data does not meet these assumptions and, e.g., only the completion times are recorded. In such cases, the researcher must make additional assumptions to apply existing algorithms. Therefore, future research on start time estimation and imputation is worth pursuing. Alternatively, existing algorithms can also be extended such that they do not longer require, e.g., explicit start timestamps.

Thirdly, most existing algorithms focus on a single modeling aspect, whereas they are in fact related to other modeling decisions. Future algorithms for supporting modeling tasks could therefore benefit from an integrated approach. For example, the resource schedules can be taken into account when mining the queue discipline as a resource might start prioritizing urgent tasks at the end of the working day.

Finally, a single analysis package should be created which takes process data as an input and provides comprehensive support for a multitude of simulation modeling decisions. This involves an integration of the current algorithms with yet to be developed techniques for other modeling decisions. Even though the development of such an integrated package does not require the development of new scientific knowledge, it is required to facilitate data-driven simulation in practice.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Decision Discovery in Business Processes](#)
- ▶ [Queue Mining](#)

References

Aguirre S, Parra C, Alvarado J (2013) Combination of process mining and simulation techniques for business process redesign: a methodological approach. *Lect Notes Bus Info Process* 162:24–43

- Baier T, Mendling J, Weske M (2014) Bridging abstraction layers in process mining. *Info Syst* 46:123–139
- Bose RPJC, van der Aalst WMP (2009) Context aware trace clustering: towards improving process mining results. In: Proceedings of the ninth SIAM international conference on data mining, pp 401–412
- Bose RPJC, van der Aalst WMP (2010) Trace clustering based on conserved patterns: towards achieving better process models. *Lect Notes Bus Info Process* 43:170–181
- Burattin A, Sperduti A, Veluscek M (2013) Business models enhancement through discovery of roles. In: Proceedings of the 2013 IEEE symposium on computational intelligence and data mining, pp 103–110
- de Leoni M, Dumas M, García-Bañuelos L (2013) Discovering branching conditions from business process execution logs. *Lect Notes Comput Sci* 7793: 114–129
- de Medeiros AKA, Guzzo A, Greco G, van der Aalst WMP, Saccà D (2008) Process mining based on clustering: a quest for precision. *Lect Notes Comput Sci* 4928:17–29
- De Weerdt J, De Backer M, Vanthienen J, Baesens B (2012) A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Info Syst* 37(7):654–676
- De Weerdt J, Vanthienen J, Baesens B, vanden Broucke SKLM (2013) Active trace clustering for improved process discovery. *IEEE Trans Knowl Data Eng* 25(12):2708–2720
- Delias P, Doumpou M, Grigoroudis E, Manolitzas P, Matsatsinis N (2015) Supporting healthcare management decisions via robust clustering of event logs. *Knowl Based Syst* 84:203–213
- Dickey D, Pearson C (2005) Recency effect in college student course evaluations. *Pract Assess Res Eval* 10(6):1–10
- Dumas M, van der Aalst WMP, Ter Hofstede AH (2005) Process-aware information systems: bridging people and software through process technology. Wiley, Hoboken
- Dumas M, La Rosa M, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer, Heidelberg
- Ferreira DR, Alves C (2012) Discovering user communities in large event logs. *Lect Notes Bus Info Process* 99:123–134
- Greco G, Guzzo A, Ponieri L, Sacca D (2006) Discovering expressive process models by clustering log traces. *IEEE Trans Knowl Data Eng* 18(8):1010–1027
- Günther CW, Rozinat A, van der Aalst WMP (2010) Activity mining by global trace segmentation. *Lect Notes Bus Info Process* 43:128–139
- Hopp WJ, Spearman ML (2011) Factory physics. Waveland Press, Long Grove
- Huang Z, Lu X, Duan H (2011) Mining association rules to support resource allocation in business process management. *Expert Syst Appl* 38(8):9483–9490
- Kelton W, Sadowski R, Zupick N (2015) Simulation with Arena. McGraw-Hill, New York

- Leyer M, Moormann J (2015) Comparing concepts for shop floor control of information-processing services in a job shop setting: a case from the financial services sector. *Int J Prod Res* 53(4):1168–1179
- Liu J, Hu J (2007) Dynamic batch processing in workflows: model and implementation. *Futur Gener Comput Syst* 23(3):338–347
- Liu Y, Wang J, Yang Y, Sun J (2008) A semi-automatic approach for workflow staff assignment. *Comput Ind* 59(5):463–476
- Liu Y, Zhang H, Li C, Jiao RJ (2012) Workflow simulation for operational decision support using event graph through process mining. *Decis Support Syst* 52(3):685–697
- Ly LT, Rinderle S, Dadam P, Reichert M (2006) Mining staff assignment rules from event-based data. *Lect Notes Comput Sci* 3812:177–190
- Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP, Toussaint J (2016) From low-level events to activities – a pattern-based approach. *Lect Notes Comput Sci* 9850:125–141
- Martin N, Bax F, Depaire B, Caris A (2016a) Retrieving resource availability insights from event logs. In: Proceedings of the 2016 IEEE international conference on enterprise distributed object computing, pp 69–78
- Martin N, Depaire B, Caris A (2016b) The use of process mining in business process simulation model construction: structuring the field. *Bus Inf Syst Eng* 58(1): 73–87
- Martin N, Depaire B, Caris A (2016c) Using event logs to model interarrival times in business process simulation. *Lect Notes Bus Inf Process* 256:255–267
- Martin N, Swennen M, Depaire B, Jans M, Caris A, Vanhoof K (2017) Retrieving batch organisation of work insights from event logs. *Decis Support Syst* 100:119–128
- Melão N, Pidd M (2003) Use of business process simulation: a survey of practitioners. *J Oper Res Soc* 54(1): 2–10
- Mărușter L, van Beest NRTP (2009) Redesigning business processes: a methodology based on simulation and process mining techniques. *Knowl Inf Syst* 21(3):267–297
- Nakatumba J (2013) Resource-aware business process management: analysis and support. Ph.D. thesis, Eindhoven University of Technology
- Nakatumba J, van der Aalst WMP (2010) Analyzing resource behavior using process mining. *Lect Notes Bus Inf Process* 43:69–80
- Nakatumba J, Westergaard M, van der Aalst WMP (2012) Generating event logs with workload-dependent speeds from simulation models. *Lect Notes Bus Inf Process* 112:383–397
- Pika A, van der Aalst WMP, Fidge CJ, ter Hofstede AHM, Wynn MT (2013) Predicting deadline transgressions using event logs. *Lect Notes Bus Inf Process* 132: 211–216
- Pospisil M, Hruška T (2012) Business process simulation for predictions. In: Proceedings of the second international conference on business intelligence and technology, pp 14–18
- Robinson S (2004) Simulation: the practice of model development and use. Wiley, Chichester
- Rogge-Solti A, Kasneci G (2014) Temporal anomaly detection in business processes. *Lect Notes Comput Sci* 8659:234–249
- Rozinat A, van der Aalst WMP (2006a) Decision mining in business processes. Tech. Rep. BPM Center Report BPM-06-10
- Rozinat A, van der Aalst WMP (2006b) Decision mining in ProM. *Lect Notes Comput Sci* 4102:420–425
- Rozinat A, Mans RS, Song M, van der Aalst WMP (2008) Discovering colored Petri nets from event logs. *Int J Softw Tools Technol Transfer* 10(1):57–74
- Rozinat A, Mans RS, Song M, van der Aalst WMP (2009) Discovering simulation models. *Info Syst* 34(3): 305–327
- Schonenberg H, Jian J, Sidorova N, van der Aalst WMP (2010) Business trend analysis by simulation. *Lect Notes Comput Sci* 6051:515–529
- Senderovich A, Weidlich M, Gal A, Mandelbaum A (2014) Mining resource scheduling protocols. *Lect Notes Comput Sci* 8659:200–216
- Song M, van der Aalst WMP (2008) Towards comprehensive support for organizational mining. *Decis Support Syst* 46(1):300–317
- Song M, Günther CW, van der Aalst WMP (2009) Trace clustering in process mining. *Lect Notes Bus Inf Process* 17:109–120
- Suriadi S, Wynn MT, Xu J, van der Aalst WMP, ter Hofstede AH (2017) Discovering work prioritisation patterns from event logs. *Decis Support Syst* 100: 77–92
- Szimanski F, Ralha CG, Wagner G, Ferreira DR (2013) Improving business process models with agent-based simulation and process mining. *Lect Notes Bus Inf Process* 147:124–138
- Tumay K (1996) Business process simulation. In: Proceedings of the 1996 winter simulation conference, pp 55–60
- van Beest NRTP, Mărușter L (2007) A process mining approach to redesign business processes – a case study in gas industry. In: Proceedings of the 2007 international symposium on symbolic and numeric algorithms for scientific computing, pp 541–548
- van der Aalst WMP (2015) Business process simulation survival guide. In: vom Brocke J, Rosemann M (eds) *Handbook on business process management*, vol 1. Springer, Heidelberg, pp 337–370
- van der Aalst WMP (2016) Process mining: data science in action. Springer, Heidelberg
- van der Aalst WMP, Nakatumba J, Rozinat A, Russell N (2010) Business process simulation. In: vom Brocke J, Rosemann M (eds) *Handbook on business process management*. Springer, Heidelberg, pp 313–338
- Veiga GM, Ferreira DR (2010) Understanding spaghetti models with sequence clustering in ProM. *Lect Notes Bus Inf Process* 43:92–103

- Vincent S (1998) Input data analysis. In: Banks J (ed) *Handbook of simulation: principles, advances, applications, and practice*. Wiley, Hoboken, pp 3–30
- Wen Y, Chen Z, Liu J, Chen J (2013) Mining batch processing workflow models from event logs. *Concurrency Comput Pract Experience* 25(13):1928–1942.
- Wombacher A, Jacob ME (2013) Start time and duration distribution estimation in semi-structured processes. In: Proceedings of the 28th annual ACM symposium on applied computing, pp 1403–1409
- Wombacher A, Jacob M, Haitsma M (2011) Towards a performance estimate in semi-structured processes. In: Proceedings of the 2011 IEEE international conference on service-oriented computing and applications, pp 1–5
- Xu R, Wunsch D (2005) Survey of clustering algorithms. *IEEE Trans Neural Netw* 16(3):645–678

Decision Discovery in Business Processes

Massimiliano de Leoni¹ and Felix Mannhardt²

¹Eindhoven University of Technology,

Eindhoven, The Netherlands

²Department of Economics and Technology Management, SINTEF Technology and Society, Trondheim, Norway

Synonyms

Decision mining; Rule mining

Definitions

In the context of business process analytics, decision discovery is the problem of discovering the set of rules that are de-facto used to make decisions in a business process, by analyzing event logs recording past executions of the process.

Overview

The execution of business processes typically follow some decisions that determine, within the multiple alternative executions, how to carry on executions of the process. These decisions are defined over the characteristics of the single

instances (e.g. loan's amount, applicant's or patient's age). This chapter reports on different approaches to discover these decision rules when unknown to process stakeholders. Most of these techniques in fact leverage on machine-learning. A number of successful real-life case-study applications are discussed along with the open-source tools that were used.

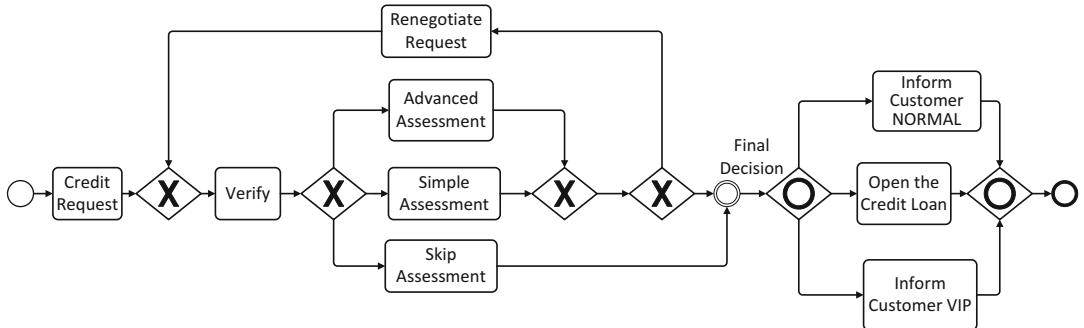
Introduction

The main focus of automatic process discovery has traditionally been in the realm of process mining on the control-flow perspective (cf. the chapter of Automatic Process Discovery). As an example, the recent advances have led to discovery algorithms that can discover models such as that in Fig. 1. This model only shows the control flow, namely, the ordering with which activities can be executed in the process.

Event logs are typically of such a form as in Fig. 2. Automatic process discovery techniques only focus on the first two columns of an event log, the case identifier (e.g., the customer id) and the activity name, thus overlooking a lot of insightful information pertaining other perspectives, such as the organization perspective, the decision perspective (a.k.a. the data or case perspective), and the time perspective (van der Aalst 2016).

The control-flow perspective is certainly of high importance as it can be considered as the process backbone; however, many other perspectives should also be considered to ensure that the model is sufficiently accurate. This chapter will focus the attention on the decision perspective and will not discuss the organization and time perspectives. Readers are referred to van der Aalst (2016) as initial pointer to explore the research results regarding the latter two perspectives.

The decision perspective focuses on how the routing of process instance executions is affected by the characteristics of the specific process instance, such as the amount requested for a loan, and by the outcomes of previous execution steps, e.g., the verification result. The representation



Decision Discovery in Business Processes, Fig. 1 A model that shows only the control-flow, i.e. the ordering of execution of activities

Customer id	Activity Name	Timestamp	Amount	Resource	Verification	Interest	Decision	Applicant	VIP
HT25	Credit Request	1-12-2012 9:00AM	3000	John	-	-	-	Max	1
HT25	Verify	3-12-2012 1:00PM	3000	Pete	OK	-	-	Max	1
HT25	Simple Assessment	7-12-2012 11:00AM	3000	Sue	OK	599	NO	Max	1
HT25	Inform Customer VIP	7-12-2012 3:00PM	3000	Mike	OK	599	NO	Max	1
HX65	Credit Request	3-12-2012 9:00AM	5000	John	-	-	-	Felix	0
HX65	Verify	5-12-2012 1:00PM	5000	Mike	NO	-	-	Felix	0
HX66	Skip Assessment	5-12-2012 2:00PM	5000	Mike	NO	-	-	Felix	0
HX65	Inform Customer NORMAL	8-12-2012 1:00PM	5000	Pete	NO	-	-	Felix	0
EA49	Credit Request	1-12-2012 9:00AM	6000	John	-	-	-	Sara	1
EA49	Verify	3-12-2012 1:00PM	6000	Pete	OK	-	-	Sara	1
EA49	Advanced Assessment	5-12-2012 1:00PM	6000	Sue	OK	1020	OK	Sara	1
EA49	Inform Customer VIP	8-12-2012 5:00PM	6000	Ellen	OK	1020	OK	Sara	1
EA49	Open the Credit Loan	8-12-2012 6:00PM	6000	Ellen	OK	1020	OK	Sara	1
HX12	Credit Request	10-12-2012 9:00AM	6000	John	-	-	-	Michael	1
HX12	Verify	13-12-2012 1:00PM	6000	Pete	OK	-	-	Michael	1
HX12	Advanced Assessment	15-12-2012 1:00PM	6000	Sue	OK	1020	NOK	Michael	1
HX12	Renegotiate Request	16-12-2012 3:00PM	6000	Sue	OK	1020	NOK	Michael	1
...

Decision Discovery in Business Processes, Fig. 2 An excerpt of an event log. Each row is an event referring to an activity that happened at a given timestamp. Several

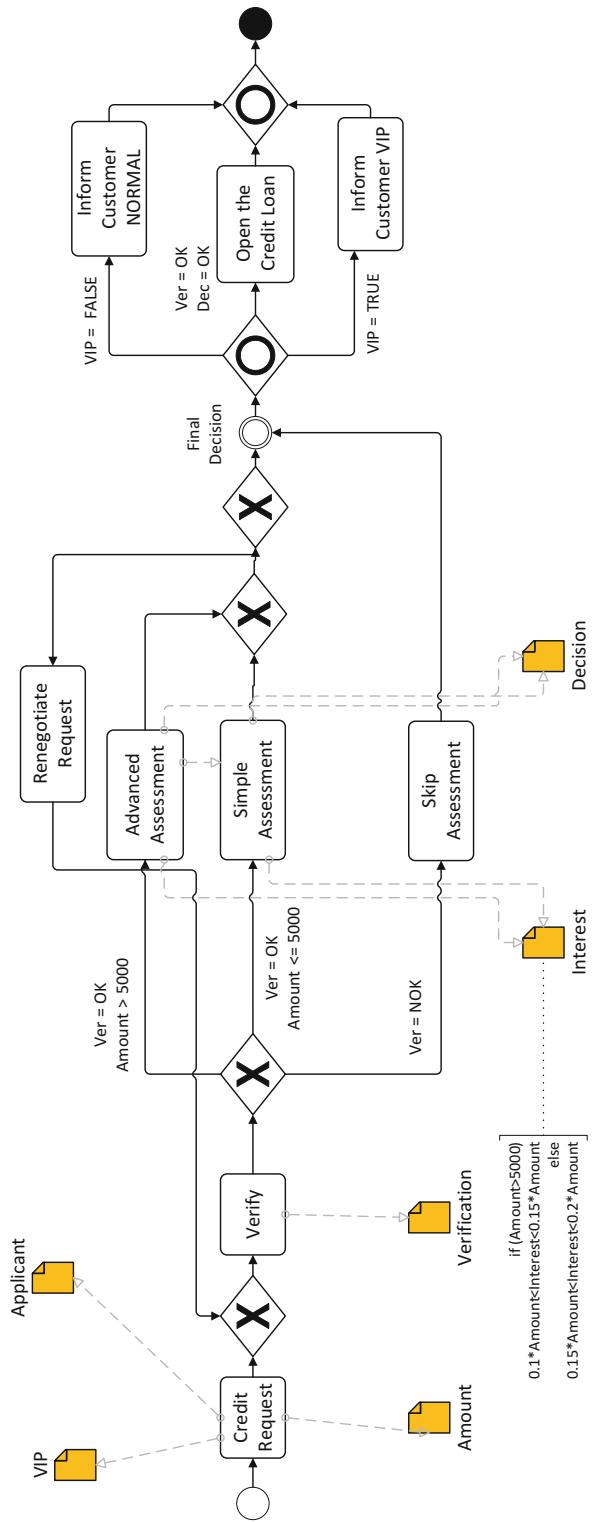
process attributes are written and updated by one or more events. The alternation of two color tones is used to group events referring to the same process instance

of this decision perspective on a process in an integrated model or as separate tables is nowadays gaining momentum. This is also testified by the recent introduction and refinement of Decision Model and Notation (DMN), which is a standard published by the Object Management Group to describe and model the decision perspective (DMN 2016).

The simplest representation of the decision perspective is to attach decision rules to XOR and OR splits, the decision points of a process model. The rules explaining the choices are driven by additional data associated with the process, which is generated by the previous process' steps. In the remainder, this additional data is abstracted as process attributes, each of which is name-value pair. Note that not all decision points are

driven by rules: the process attributes are not able to determine which branch is going to be followed (cf. the XOR split involving *Renegotiate Request*). Figure 3 illustrates a possible extension of the model in Fig. 1 to incorporate the data attributes that are manipulated by the process and how they affect the routing at the decision points. The rules at decision points can be alternatively easily represented as a separate DMN table.

Historically, the discovery of the decision perspective is called *Decision Mining*, a name that was introduced by the seminal work of Rozinat and van der Aalst (2006). In this work, the mined decisions at the decision points are mutually exclusive: when a process instance reaches a decision point, one and exactly one branch is enabled. Rozinat et al. leverage on Petri nets, but



Decision Discovery in Business Processes, Fig. 3 A model that extends what is depicted in Fig. 1. The decision points are annotated with rules enabling the branches

the same idea can trivially be moved to BPMN and other process modeling notations. In this work, decision-mining problems are transformed into classification problems: techniques such as decision-tree learning can be leveraged off. Decision trees are learnt from an observation instance table. For each decision point in the process, the subset of the events that refers to the activities that follow the decision point is extracted. Each observation instance consists of a dependent variable, a.k.a. response variable, and a set of independent variables, a.k.a. predictor variables, that are used for prediction. The dependent variable is the activity of the event following, and (a subset of) the event (i.e., process) attributes are used as independent variables. This chapter denotes variables as the features used in observation instances to learn a (decision-tree) classifier. Attributes are conversely associated with processes and events and indicate the data points manipulated while executing a process instance.

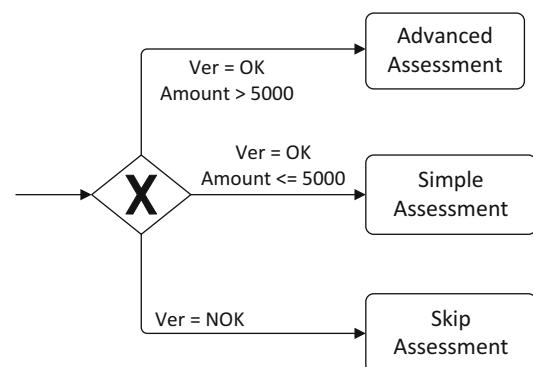
Decision Mining as a Classification Problem

The introduction has illustrated that the problem of decision mining can be translated into a classification problem. Several classification techniques exist; however, one needs to focus on those techniques that return human-readable decision rules in the form of, e.g., formulas. This motivates why most of the decision-mining techniques, such as the seminar work (Rozinat and van der Aalst 2006) and others (Batoulis et al. 2015; de Leoni and van der Aalst 2013; Bazhenova et al. 2016), aim to learn decision trees and use them to build a set of decision rules. In addition, to provide human-readable rules, decision trees have the advantage of creating rules that are clearly highlighting the attributes whose values affect the decisions. These works typically leverages on the C4.5 algorithm for decision-tree learning, which is capable to deal with noise and missing values. In the context of decision mining, noise indicates that in a fraction of situations, the wrong (or unusual) decision is made; missing values indicate that, due to logging errors, cer-

tain assignments of values to attributes are not recorded or, also, process participants forgot to update the value of given attributes. Association rule discovery techniques might be a possibility, but they tend to return a large number of rules that do not necessarily have a discriminatory function on the decision, either.

Let us consider the decision point in Fig. 4, part of the model in Fig. 3. Clearly, the rules are initially not present and the aim is to discover them. From the event log, every event that refers to the activities at the decision point is retained from the event log. Considering the event log in Fig. 2, this results in an observation instance table such as in Fig. 5. The column *Activity Name* refers to the dependent variable, and the other columns are the independent variables, categorical or numerical, that are used to predict the dependent variable. The application of decision-tree learning techniques will produce a tree similar to that in Fig. 6.

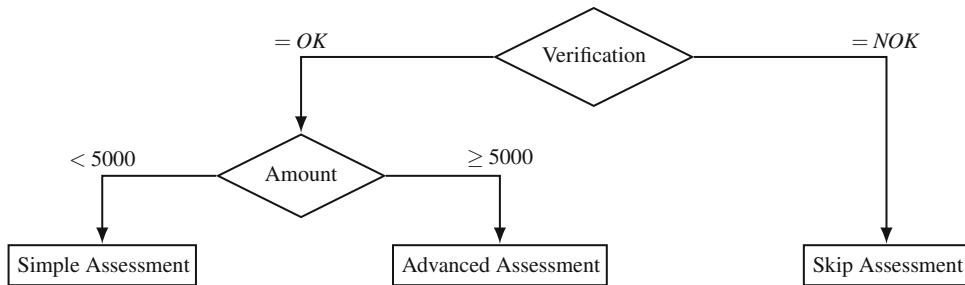
Decision trees classify instances by sorting them down in a tree from the root to some leaf node. Each non-leaf node specifies a test of some variable x_1, \dots, x_n (in decision mining, the process attributes), and each branch descending from that node corresponds to a range of possible values for this variable. In general, a decision tree represents a disjunction of conjunctions of expressions: each path from the tree root to a leaf corresponds to an expression that is, in fact, a conjunction of variable tests. Each leaf node is



Decision Discovery in Business Processes, Fig. 4 One of the decision points. The aim is to start from the control flow and discover the rules

Customer id	Activity Name	Timestamp	Amount	Resource	Verification	Interest
HT25	Simple Assessment	7-12-2012 11:00AM	3000	Sue	OK	599
HX66	Skip Assessment	5-12-2012 2:00PM	5000	Mike	NO	-
EA49	Advanced Assessment	5-12-2012 1:00PM	6000	Sue	OK	1020
HX12	Advanced Assessment	15-12-2012 1:00PM	6000	Sue	OK	1020
...

Decision Discovery in Business Processes, Fig. 5 An excerpt of the observation instances that are extracted from the event log in Fig. 2 to discover the rules at the decision point in Fig. 4



Decision Discovery in Business Processes, Fig. 6 A possible decision tree discovered based on the observation instances of which an excerpt is in Fig. 5

associated with one of the possible values of the dependent variables, namely, with one of the process activities that follows an XOR split: if an expression e is associated with a path to a leaf node a , every input tuple for which e evaluates to true is expected to return a as output. In decision mining, this means that every time a decision point is reached and the process attributes x_1, \dots, x_n take on values that make e evaluate true, the process execution is expected to continue with activity a . In the example tree in Fig. 6, each leaf refers to a different activity. However, multiple leaves can be associated with the same activity a . In that case, each path leading to a would correspond to a different expression. Together all these expressions combined with disjunction (i.e., a logical OR operator) constitute an expression that predicts when the process continues with activity a . As discussed in van der Aalst (2016, pp. 294–296), this procedure is extensible to OR splits, where multiple branches may be activated.

Extension of the Basic Technique

This basic technique has been extended to deal with additional cases. The first extension is re-

lated to the situations when the executions are not always compliant with the control-flow model and/or the model contains invisible steps. The second extension is applicable when the rules at decision points are not mutually exclusive, such as when some information that drives the decisions is missing or the process behavior is simply not fully deterministic. These extensions are discussed in detail below.

There are other valuable extensions, which cannot be elaborated on in detail for the sake of space. In Bazhenova et al. (2016) and De Smedt et al. (2017b), authors extend the basic algorithm to be able to discover the dependencies between process attribute updates and leverage on invariant miners. For example, the values assignable to an attribute *Risk* are a function of the values taken on by attributes *Amount* and *Premium*. While the basic BPMN modeling notation is only able to represent path routing decisions, BPMN can be complemented with DMN where data-related decisions can be represented. As discussed in Batoulis et al. (2015), DMN is especially relevant when models are of significant size. By complementing BPMN with DMN, the decisions are modeled outside the control-flow model. The paper argues that this separation of concerns would

Customer id	Activity Name	Timestamp	Amount	Resource	Verification
TH25	Credit Request	1-11-2012 9:00AM	4000	John	-
TH25	Simple Assessment	7-11-2012 11:00AM	4000	Sue	-
TH25	Inform Customer VIP	7-11-2012 3:00PM	4000	Mike	-
TH26	Credit Request	1-1-2013 19:00AM	4100	Sue	-
TH26	Simple Assessment	1-2-2013 11:00AM	4100	John	-
TH26	Verify	3-2-2013 15:00PM	4100	Mike	NOK
TH26	Inform Customer NORMAL	17-2-2012 3:00PM	4100	Pete	NOK
...

Decision Discovery in Business Processes, Fig. 7 Two non-compliant traces for the model in Fig. 1: they break the assumption of the basic decision-tree algorithm

increase the readability and the maintainability of these models. Here, maintainability of a model is intended as the easiness of extending, adjusting, and updating a model when the internal protocols or the external rules and regulations change.

Non-compliance and Invisible Steps

The first assumption of the basic algorithm is that, when discovering the rules of a decision point, every activity is always executed in the right moment according to the dictation of the control-flow model, e.g., the model in Fig. 1. However, due to logging errors and non-compliances, sometimes executions do not comply with the control-flow model. As an example, consider trace for the customer with id *TH25* in Fig. 7. This trace is clearly not compliant with the model in Fig. 1: activity *Credit Request* is immediately followed by *Simple Assessment*, i.e., activity *Verify* did not occur (or its execution was not logged). This means that the value of attribute *Verification* is missing. Several decision-tree learning algorithms, such as C4.5, are very good at dealing with missing values. However, it is necessary to detect that a value is missing due to non-compliances. As an example, the execution for the customer with id *TH25* and the decision of executing *Simple Assessment* were made without considering the value of attribute *Verification*; in fact, the value of that attribute was undefined because the execution of activity *Verification* never took place. If such cases as this are not detected, one mistakenly could use an old value. For instance, attribute *Verification* is updated multiple times via multiple executions

of activity *Verification*, which is executed once for each loan's renegotiation. If one renegotiation mistakenly skipped the verification, decision mining might use the previous value, updated during the previous renegotiation's loop. The basic algorithm also suffers of the problem that one should ignore updates of attributes that result from the non-compliant execution of activities. As an example, consider trace for the customer with id *TH26* in Fig. 7: activity *Verify* occurs after *Simple Assessment*, which violates the constraints dictated by the model in Fig. 1. As a result of the execution of this activity, attribute *Verification* takes on value *NOK* that is too late because a simple assessment has already been conducted. It is clear that, in this case, activity *Simple Assessment* should not have occurred.

To detect missing values and illegal attribute updates, the work report in de Leoni and van der Aalst (2013) integrates the basic technique with techniques for log-model alignment. An alignment between a recorded process execution (log trace) and a process model is a pairwise matching between activities recorded in the log and activities allowed by the model.

Table 1 illustrates the alignments of the model in Fig. 1 with the traces in Fig. 7. Activity names are abbreviated with the first letter of each name word. Abstracting from the attribute updates and ignoring \gg 's, for each log trace, the log row corresponds to the log trace, and the process row represents the execution allowed by the process model that is the closest to the log traces in terms of number of necessary changes. The yellow and green columns refer to log and model moves, respectively. For further information, readers are

referred to Adriansyah (2014) and to the chapter on Conformance Checking. The events for log moves will be ignored for decision mining. Conversely, activities related to model moves are introduced and will be considered. Attributes that are supposedly updated as a result of the activity executions take on null, a special value to indicate that the value is missing and should be treated as such by the decision-tree learning algorithm. An alternative to using alignments could be to just remove the traces referred to executions not compliant with the control-flow model. However, situations in which most of the traces are almost compliant and few are fully compliant would cause the rules to be mined on the basis of insufficient observation instances. If the missing events refer to activities that do not update attributes relevant for a certain decision point, there would be no reason to discard the corresponding traces when discovering the rules for such a decision point.

In addition to dealing with non-compliance, de Leoni and van der Aalst (2013) allows for

Decision Discovery in Business Processes, Table 1
Alignments of the traces in Fig. 7 wrt. the model in Fig. 1

Customer with id TH25

Log:	CR	»	SA	ICV
Process:	CrR	V	SA	ICV

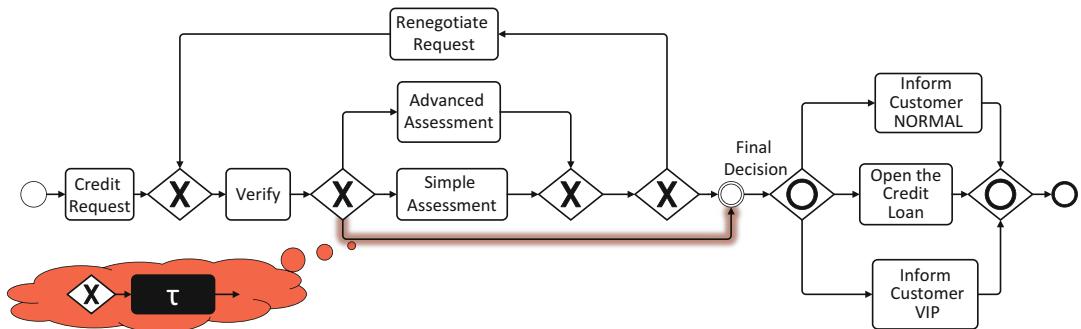
Customer with id TH26

Log:	CR	SA	V	»	ICN
Process:	CR	»	V	SA	ICN

discovering decision rules for models such as in Fig. 8. In this case, the explicit activity *Skip Assessment*, which indicates that no activity is performed in case of negative verification but, conversely, the execution directly moves to the negative final decision, is not present and recorded in the respective traces. In such a case, no activity can be explicitly identified to indicate that the assessment does not take place. To overcome this, the approach in de Leoni and van der Aalst (2013) would assume that there is a silent activity τ , as illustrated by the cloud in the figure. Using alignments, silent activity τ would be included in the analysis. When an execution of τ is necessary (e.g., when the assessment is skipped), the alignment would automatically add a model move for τ , and as mentioned above, model moves are always considered when mining the decision rules. Of course, the step τ is always involved in model moves since they are never recorded in the event log, being indeed silent.

Overlapping Rules

Existing decision-mining techniques for exclusive choices rely on the strong assumption that the rules attached to the alternative activities of an exclusive choice need to be mutually exclusive. However, business rules are often nondeterministic, and this “cannot be solved until the business rule is instantiated in a particular situation” (Rosca and Wild 2002). This ambiguity can occur due to conflicting rules or missing contextual information. For example, decisions taken by process workers may depend on contextual



Decision Discovery in Business Processes, Fig. 8 A variation of the model in Fig. 1: activity *Skip Assessment* is removed. Ideally, it can be replaced by a silent activity

τ , whose execution is not recorded in any trace of the event log

factors, which are not encoded in the system and, thus, not available in event logs (Bose et al. 2013). In Mannhardt et al. (2016), a technique is proposed that discovers overlapping rules in those cases that the underlying observations are characterized better by such rules. The technique is able to deliberately trade the precision of mutually exclusive rules, i.e., only one alternative is possible, against fitness, i.e., the overlapping rules that are less often violated. In short, as in the basic algorithm, the technique builds an initial decision tree based on observations from the event log, and for each activity a , a decision rule $rule1(a)$ is found (possibly $rule1(a) = \text{true}$). Then, for each decision-tree leaf, the wrongly classified instances are used to learn a new decision tree leading to new rules, such as $rule2(a)$. These new rules are used in disjunction with the initial rules yielding, e.g., overlapping rules of the form $rule1(a) \vee rule2(a)$.

It is worth highlighting here that overlapping rules are appropriate in contexts where information is missing and/or rules are intrinsically not fully deterministic. However, in other contexts, overlapping rules are just the results of wrong decisions made by process modelers and experts. In those cases, overlapping rules should be prevented and, when present, be repaired to ensure them to be mutual exclusive. Calvanese et al. (2016) propose a technique to detect overlapping rules and fix them to ensure their mutual exclusiveness.

Decision-Aware Control-Flow Discovery

De Smedt et al. (2017a) distinguish two categories of decision-mining techniques. The approaches that fall into the first category are named *decision-annotated process mining*: first a suitable control-flow model is discovered, which is later annotated with decision rules. Every approach discussed so far is within this category. A second category aims at *decision-aware control-flow discovery*: the control-flow and decisions are discovered together in a holistic manner.

Any approach in the first category has the disadvantage that it “*imposes the initial struc-*

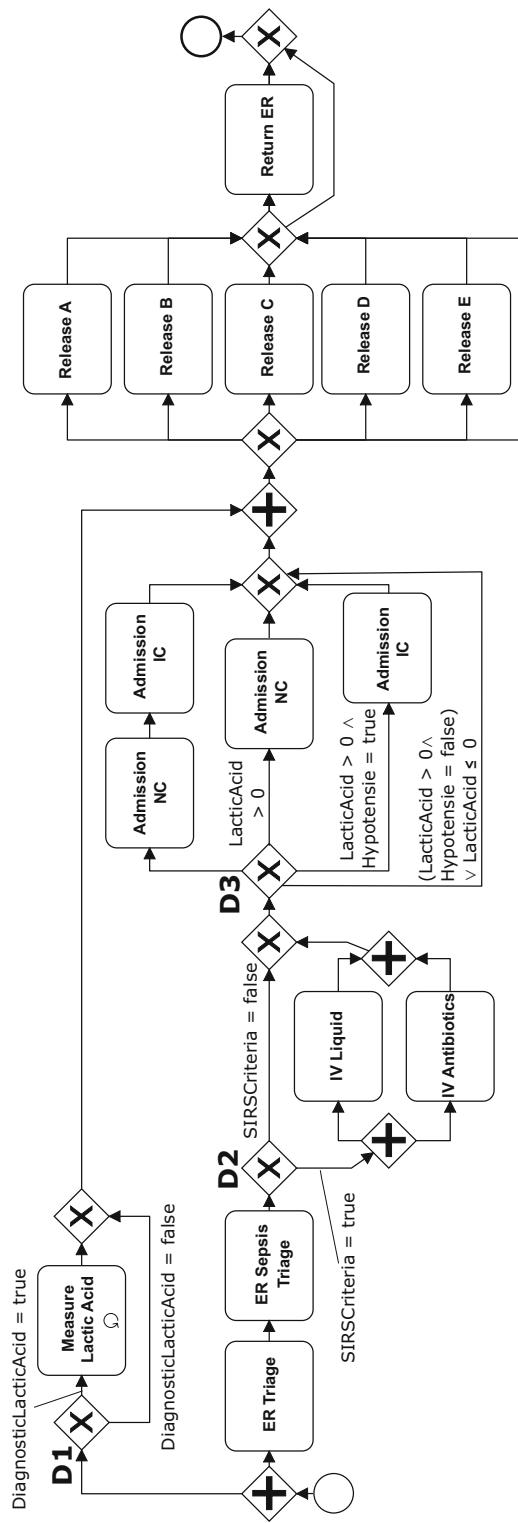
ture on top of which the decision perspective is placed upon” (De Smedt et al. 2017a). This might cause important decisions to remain unrevealed: the control-flow structure is discovered without considering the overall logic of the decisions, and important decision points are not made explicit.

Whereas Maggi et al. (2013) and Schönig et al. (2016) report on works to simultaneously discover data- and control-flow for declarative models (cf. the chapter of Declarative Modelling), Mannhardt et al. (2017) report the only research work about decision-aware control-flow discovery of procedural models (i.e., BPMN-like) where the decision perspective is used to reveal paths in the control flow that can be characterized by deterministic rules over the recorded attributes. If such behavior is infrequently observed, it is often disregarded by control-flow discovery techniques as noise (cf. the chapter on automated process discovery). The idea is that some paths may be executed infrequently because the corresponding conditions are rarely fulfilled. These paths and the conditions are likely to be of great interest to process analysts (e.g., in the context of risks and fraud). In Mannhardt et al. (2017), classification techniques are employed to distinguish between such behavior and random noise.

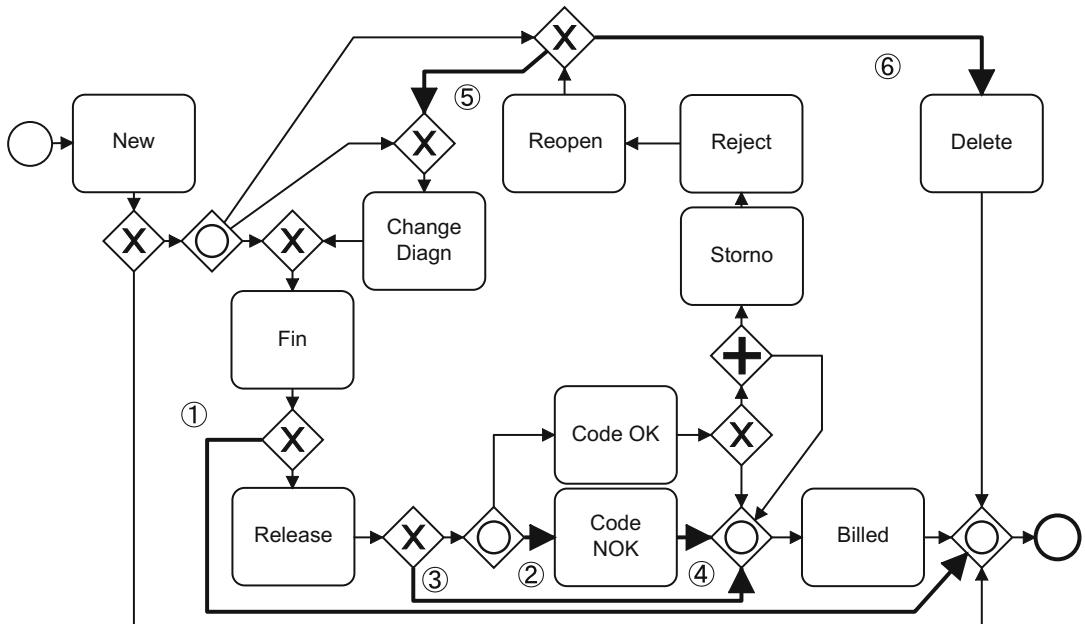
Example Cases and Tool Support

Decision mining has been applied in several domains as testified by many applications reported in literature (Rozinat 2010; de Leoni and van der Aalst 2013; Mannhardt et al. 2016, 2017; De Smedt et al. 2017b; Bazhenova and Weske 2016; Mannhardt 2018) and several implemented tools (van Dongen et al. 2005; Mannhardt et al. 2015; Kalenkova et al. 2014; Bazhenova et al. 2017). Here, two example cases are briefly presented. Both cases are taken from a project that was conducted in a regional hospital in the Netherlands (Mannhardt 2018).

In the first case, the trajectories of patients that are admitted to the *emergency ward* of the hospital with a suspicion for sepsis (Rhodes et al. 2017) were analyzed. A normative model of the expected patient trajectory control flow was de-



Decision Discovery in Business Processes, Fig. 9 Overlapping and nonoverlapping decision rules discovered for the trajectory of patients in a Dutch hospital. This model is an excerpt from the model in Mannhardt et al. (2016) and Mannhardt (2018)



Decision Discovery in Business Processes, Fig. 10 Process model discovered by the decision-aware process discovery method (Mannhardt et al. 2017) for a hospital billing process (Mannhardt et al. 2017)

signed and, then, the overlapping decision rules method (Mannhardt et al. 2016), yielding the model in Fig. 9.

In the second case, the hospital's billing process for medical services was analyzed. Several medical services (e.g., diagnostics, treatments, etc.) are collected in a billing package and validated and invoiced. However, in some cases, the billing package is *reopened*, *rejected*, or *canceled*, and further changes are needed. Figure 10 shows a process model discovered by the decision-aware control-flow discovery method. Six infrequent control-flow paths that are associated with specific data attribute values have been included in the process model. Some of these paths should not be disregarded as noise since they are related to cases with rework. For example, the path between RELEASE and Code NOK (②) occurs mostly for two specific *caseType* values. According to a domain expert, both case types correspond to exceptional cases: one is used for intensive care and the other for cases in which the code cannot be obtained (Code NOK). Another example is the path from Code NOK to Billed (④), which is also related to

the *caseType* attribute as well as to the medical *specialty* attribute. This path is interesting since it could not be explained by the domain expert.

Conclusion

This chapter has discussed the importance of the *decision perspective* and illustrates several techniques that aim to explain how the choices are driven by additional data associated with the process. These techniques leverage on the additional information recorded in data attributes (also denoted as event payload) of the event log to discover process models and enhance existing ones so that the models explain the decisions that drive the process executions. It was illustrated that these choices can be represented as annotations of process models or, alternatively, the decision can be modeled through separate DMN models that highlight the decision-related aspects.

As discussed, the application to real-life cases has illustrated that the maturity of the decision-mining approach has reached a level that certainly allows its application to production

environments. While it is certainly necessary to extend the class of decisions that can be mined, the major drawback is that the decisions are mining locally without considering the other decisions discovered at different points. This means that decision rules at different decision points can be conflicting, leading to models that, if actually enacted, would cause a certain number of executions to be unable to properly terminate. As future work, it is crucial to check whether the models enriched with decision rules are correct and to ensure to some extend that decisions are not conflicting, thus leading to deadlocks.

Cross-References

- [Automated Process Discovery](#)
- [Conformance Checking](#)

References

- Adriansyah A (2014) Aligning observed and modeled behavior. PhD thesis, Eindhoven University of Technology. <https://doi.org/10.6100/IR770080>
- Batoulis K, Meyer A, Bazhenova E, Decker G, Weske M (2015) Extracting decision logic from process models. In: CAiSE 2015. LNCS, vol 9097. Springer, pp 349–366. https://doi.org/10.1007/978-3-319-19069-3_22
- Bazhenova E, Weske M (2016) Deriving decision models from process models by enhanced decision mining. In: BPM 2015 workshops. Springer, pp 444–457. https://doi.org/10.1007/978-3-319-42887-1_36
- Bazhenova E, Bülow S, Weske M (2016) Discovering decision models from event logs. In: BIS 2016. LNBIP, vol 255. Springer, pp 237–251. https://doi.org/10.1007/978-3-319-39426-8_19
- Bazhenova E, Haarmann S, Ihde S, Solti A, Weske M (2017) Discovery of fuzzy DMN decision models from event logs. In: CAiSE 2017. LNCS, vol 10253. Springer, pp 629–647. https://doi.org/10.1007/978-3-319-59536-8_39
- Bose JC, Mans RS, van der Aalst WMP (2013) Wanna improve process mining results? In: CIBM 2013. IEEE, pp 127–134. <https://doi.org/10.1109/cidm.2013.6597227>
- Calvanese D, Dumas M, Ülari Laurson, Maggi FM, Montali M, Teinemaa I (2016) Semantics and analysis of DMN decision tables. In: BPM 2016. LNCS, vol 9850. Springer, pp 217–233. https://doi.org/10.1007/978-3-319-45348-4_13
- de Leoni M, van der Aalst WMP (2013) Data-aware process mining: discovering decisions in processes using alignments. In: SAC 2013. ACM, pp 1454–1461. <https://doi.org/10.1145/2480362.2480633>
- De Smedt J, vanden Broucke SKLM, Obregon J, Kim A, Jung JY, Vanthienen J (2017a) Decision mining in a broader context: an overview of the current landscape and future directions. In: BPM 2016 workshops. LNBIP, vol 281. Springer, pp 197–207. https://doi.org/10.1007/978-3-319-58457-7_15
- De Smedt J, Hasic F, vanden Broucke S, Vanthienen J (2017b) Towards a holistic discovery of decisions in process-aware information systems. In: BPM 2017. LNCS, vol 10445. Springer. https://doi.org/10.1007/978-3-319-65000-5_11
- DMN (2016) Decision Model and Notation (DMN) v1.1. <http://www.omg.org/spec/DMN/1.1/>
- Kalenkova AA, de Leoni M, van der Aalst WMP (2014) Discovering, analyzing and enhancing BPMN models using prom. In: BPM 2014 Demos, CEUR-WS.org, CEUR workshop proceedings, vol 1295, p 36
- Maggi FM, Dumas M, García-Bañuelos L, Montali M (2013) Discovering data-aware declarative process models from event logs. In: BPM 2013. LNCS, vol 8094. Springer, pp 81–96. https://doi.org/10.1007/978-3-642-40176-3_8
- Mannhardt F (2018) Multi-perspective process mining. PhD thesis, Eindhoven University of Technology
- Mannhardt F, de Leoni M, Reijers HA (2015) The multi-perspective process explorer. In: BPM 2015 Demos, CEUR-WS.org, CEUR workshop proceedings, vol 1418, pp 130–134
- Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP (2016) Decision mining revisited – discovering overlapping rules. In: CAiSE 2016. LNCS, vol 9694. Springer, pp 377–392. https://doi.org/10.1007/978-3-319-39696-5_23
- Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP (2017) Data-driven process discovery – revealing conditional infrequent behavior from event logs. In: CAiSE 2017. LNCS, vol 10253, pp 545–560. https://doi.org/10.1007/978-3-319-59536-8_34
- Rhodes A et al (2017) Surviving sepsis campaign: international guidelines for management of sepsis and septic shock: 2016. Intensive Care Med 43(3):304–377. <https://doi.org/10.1007/s00134-017-4683-6>
- Rosca D, Wild C (2002) Towards a flexible deployment of business rules. Expert Syst Appl 23(4):385–394. [https://doi.org/10.1016/s0957-4174\(02\)00074-x](https://doi.org/10.1016/s0957-4174(02)00074-x)
- Rozinat A (2010) Process mining: conformance and extension. PhD thesis, Eindhoven University of Technology, Eindhoven
- Rozinat A, van der Aalst WMP (2006) Decision mining in ProM. In: BPM 2006. LNCS, vol 4102. Springer, pp 420–425. https://doi.org/10.1007/11841760_33
- Schönig S, Di Ciccio C, Maggi FM, Mendling J (2016) Discovery of multi-perspective declarative process models. In: ICSOC 2016. LNCS, vol 9936. Springer, pp 87–103. https://doi.org/10.1007/978-3-319-46295-0_6

van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer. <https://doi.org/10.1007/978-3-662-49851-4>

van Dongen BF, de Medeiros AKA, Verbeek HMW, Weijters AJMM, van der Aalst WMP (2005) The prom framework: a new era in process mining tool support. In: ICATPN 2005. LNCS, vol 3536. Springer, pp 444–454

Decision Mining

► [Decision Discovery in Business Processes](#)

Decision Support Benchmarks

► [Analytics Benchmarks](#)

Declarative Process Mining

Fabrizio Maria Maggi
University of Tartu, Tartu, Estonia

Synonyms

[Constraint-based process mining](#); [Rule-oriented process mining](#)

Definitions

In this chapter, we introduce the techniques existing in the literature in the context of declarative process mining, i.e., process mining techniques based on declarative process modeling languages. A declarative process mining technique is any technique that, in addition to taking as input an event log, takes as input or as output a (business) process model represented in a declarative process modeling notation. Declarative process mining techniques include techniques for process

discovery, conformance checking, and compliance monitoring.

Introduction

Process discovery, conformance checking, and process model enhancement are the three basic forms of process mining (van der Aalst 2016). In particular, process discovery aims at producing a process model based on example executions in an event log and without using any a priori information. Conformance checking pertains to the analysis of the process behavior as recorded in an event log with respect to an existing reference model. Process model enhancement aims at enriching and adapting existing process models with information retrieved from logs.

In the last years, several works have been focused on process mining techniques based on declarative process models (Maggi et al. 2011a, 2012a; de Leoni et al. 2012a; Di Cicco and Mecella 2013; vanden Broucke et al. 2014; Schönig et al. 2016b; Burattin et al. 2016). The dichotomy procedural versus declarative can be seen as a guideline when choosing the most suitable language to represent models in process mining algorithms: process mining techniques based on procedural languages can be used for predictable processes working in stable environments, whereas techniques based on declarative languages can be used for unpredictable, variable processes working in turbulent environments (Zugal et al. 2011; Pichler et al. 2011; Haisjackl et al. 2013; Reijers et al. 2013; Silva et al. 2014).

Indeed, a declarative approach allows the business analyst to focus on the formalization of few relevant and relatively stable-through-time constraints that the process should satisfy, avoiding the burden to model many process control flow details that are fated to change through time. Using declarative formalisms the processes are kept under-specified so that few rules allow for multiple execution paths. In this way, the process representation is more robust to changeable behaviors and, at the same time, remains compact. For these reasons, declarative process mining

techniques have been largely applied in the context of healthcare processes (Grando et al. 2012, 2013; Rovani et al. 2015), since these processes are characterized by a high degree of variability.

Since declarative process modeling languages are focused on ruling out forbidden behavior, these languages are very suitable for defining compliance models that are used for checking that the behavior of a system complies certain regulations. The compliance model defines the rules related to a single instance of a process, and the expectation is that all the instances follow the model. Processes can be diagnosed for compliance violations in an a posteriori or offline manner, i.e., after process instance execution has been finished (conformance checking). However, the progress of a potentially large number of process instances can be monitored at runtime to detect or even predict compliance violations. For this, typically, terms such as compliance monitoring or online auditing are used (Ly et al. 2013, 2015).

In van der Aalst et al. (2009), Westergaard and Maggi (2011a), and Pesic et al. (2007), the authors introduce a declarative process modeling language called DECLARE with formal semantics grounded in LTL (linear temporal logic) (Pnueli 1977). A DECLARE model is a set of DECLARE constraints defined as instantiations of DECLARE templates that are parameterized classes of rules.

The bulk of the declarative process mining approaches proposed in the recent years are based on DECLARE. In the following sections, we give an overview of these approaches grouping them into three categories, i.e., approaches for process discovery, conformance checking, and compliance monitoring. Some of these approaches have been implemented in the process mining toolset ProM (Maggi 2013).

Process Discovery

In the area of process discovery, Lamma et al. (2007a, b) and Chesani et al. (2009a) describe the usage of inductive logic programming techniques to mine models expressed as a SCIFF (Alberti et al. 2008) first-order logic theory. Bel-

lodi et al. (2010a, b) extend this technique by weighting in a second phase the constraints with a probabilistic estimation. In particular, the learned constraints are translated into Markov Logic formulae that allow for a probabilistic classification of the traces. Both the techniques in Lamma et al. (2007a) and Bellodi et al. (2010a) rely on the availability of positive and negative traces of execution.

Maggi et al. (2011b) first proposed an unsupervised algorithm for the discovery of declarative process models expressed using DECLARE. This approach was improved in Maggi et al. (2012a) using a two-phase approach. The first phase is based on an a priori algorithm used to identify frequent sets of correlated activities. A list of candidate constraints is built on the basis of the correlated activity sets. In the second phase, the constraints are checked by replaying the log on specific automata, each accepting only those traces that are compliant to one constraint. Those constraints satisfied by a percentage of traces higher than a user-defined threshold are discovered.

Other variants of the same approach are presented in Maggi et al. (2013a), Bernardi et al. (2014b), Maggi (2014), Kala et al. (2016), and Maggi et al. (2018). The technique presented in Maggi et al. (2013a) leverages a priori knowledge to guide the discovery task. In Bernardi et al. (2014b), the approach is adapted to be used in cross-organizational environments in which different organizations execute the same process in different variants. In Maggi (2014), the author extends the approach to discover metric temporal constraints, i.e., constraints taking into account the time distance between events. Finally, in Kala et al. (2016) and Maggi et al. (2018), the authors propose mechanisms to reduce the execution times of the original approach presented in Maggi et al. (2012a).

In Di Ciccio and Mecella (2012, 2013, 2015), the authors introduce MINERful, a two-step algorithm for the discovery of DECLARE constraints. The first step of the approach is the building of a knowledge base, with information about temporal statistics about the (co-)occurrence of tasks within the log. Then, the validity and the support

of constraints are computed by querying that knowledge base. The value assigned to support is calculated by counting the activations not leading to violations of constraints. In Di Ciccio et al. (2014, 2016), the authors propose an extension of MINERful to discover target-branched DECLARE constraints, i.e., constraints in which the target parameter is replaced by a disjunction of actual tasks.

Another approach for the discovery of DECLARE models is described in Schönig et al. (2016b). The presented technique is based on the translation of DECLARE templates into SQL queries on a relational database instance, where the event log has previously been stored. The query answer assigns the free variables with those tasks that lead to the satisfaction of the constraint in the event log. The methodology has later been extended toward multi-perspective DECLARE discovery (Schönig et al. 2016a), to include data in the formulation of constraints.

Beforehand, an approach for discovering data-aware constraints was proposed in Maggi et al. (2013c). Recently, approaches for taking into account activity life cycles in the discovery task have been proposed in Bernardi et al. (2014a, 2016).

Another approach for the discovery of DECLARE models was presented in vanden Broucke et al. (2014). Here, the authors present the Evolutionary Declare Miner that implements the discovery task using a genetic algorithm.

Approaches for the online discovery of DECLARE models have been presented in Maggi et al. (2013b) and Burattin et al. (2014, 2015). Here, the models are discovered from streams of events generated at runtime during the execution of a business process. The models are adapted on the fly as soon as the behavior of the underlying process changes.

An algorithm for the discovery DCR graphs has been proposed in Debois et al. (2017). In Ferilli (2014), the authors introduce the WoMan framework that includes a method for discovering first-order logic constraints from event logs. It guarantees incrementality in learning and adapting the models and the ability to express triggers and conditions on the process tasks.

Finally, we remark that studies have been conducted to remove inconsistencies and redundancies from discovered declarative models (Di Ciccio et al. 2015, 2017). The proposed solutions resort on the language inclusion and cross-product of the automata underlying the constraints. Approaches for ensuring the relevance of a set of DECLARE constraints discovered from an event log in terms of non-vacuity to the log have been presented in Maggi et al. (2016) and Di Ciccio et al. (2018).

D

Conformance Checking

In recent years, an increasing number of researchers are focusing on the conformance checking with respect to declarative models. For example, in Chesani et al. (2009b), an approach for compliance checking with respect to *reactive business rules* is proposed. Rules, expressed using DECLARE, are mapped to abductive logic programming, and Prolog is used to perform the validation. The approach has been extended in Montali et al. (2010), by mapping constraints to LTL and evaluating them using automata. The entire work has been contextualized into the service choreography scenario.

In Burattin et al. (2012), the authors report an approach that can be used to evaluate the conformance of a log with respect to a DECLARE model. In particular, their algorithms compute, for each trace, whether a DECLARE constraint is violated or fulfilled. Using these statistics the approach allows the user to evaluate the *healthiness* of the log. The approach is based on the conversion of DECLARE constraints into automata, and using so-called activation trees, it is able to identify violations and fulfillments.

The work described in de Leoni et al. (2012b, 2014) consists in converting a DECLARE model into an automaton and perform conformance checking of a log with respect to the generated automaton. The conformance checking approach is based on the concept of alignment, and as a result of the analysis each trace is converted into the most similar trace that the model accepts. Other approaches for trace alignment

with respect to LTL specifications have been presented in De Giacomo et al. (2016, 2017). Here, automated planning is used to improve the performance of the alignment task.

In Grando et al. (2012, 2013), the authors use DECLARE to model medical guidelines and to provide semantic (i.e., ontology-based) conformance checking measures. In a recent work, reported in Borrego and Barba (2014), the data perspective for conformance checking with DECLARE is expressed in terms of conditions on global variables disconnected from the specific DECLARE constraints expressing the control flow.

In Westergaard and Maggi (2012) and Maggi and Westergaard (2014), the authors propose an approach for checking DECLARE constraints augmented with temporal conditions. In Burattin et al. (2016), a conformance checking technique for multi-perspective declarative process models is presented. In particular, in this work, the authors first describe and formalize MP-DECLARE (i.e., a multi-perspective semantics of DECLARE augmented with data and time constraints). Then, they provide an algorithmic framework to efficiently check the conformance of MP-DECLARE with respect to event logs.

One of the most well-known tools for conformance checking with declarative specifications is the *LTL Checker* presented for the first time in van der Aalst et al. (2005). This tool allows the user to check the validity of a declarative specification expressed in terms of LTL rules in the traces of a log.

Compliance Monitoring

Several approaches have been presented for compliance monitoring of business processes with respect to a set of compliance rules. For example, MobiConLTL (Maggi et al. 2011a,c) is a compliance monitoring tool implemented as a provider of the operational support in ProM (Westergaard and Maggi 2011b; Maggi and Westergaard 2017; Maggi et al. 2012b). It takes as input a reference model expressed in the form of DECLARE rules. More generally, every business constraint

that can be expressed as an LTL formula can be monitored using MobiConLTL. At runtime, a stream of events can be sent by an external workflow management system to MobiConLTL through the operational support. The provider returns rich diagnostics about the status of each rule in the reference model with respect to the traces included in the stream. This approach has been extended in De Masellis et al. (2014) to monitor data-aware DECLARE constraints and in De Giacomo et al. (2014) to cover a larger set of rules expressed using LDL (De Giacomo and Vardi 2013).

MobiConEC is a compliance monitoring framework based on a reactive version (Bragaglia et al. 2012) of the event calculus (Kowalski and Sergot 1986). Being based on the event calculus, it can easily accommodate explicit (metric and qualitative) time constraints, as well as data-enriched events and corresponding data-aware conditions. In particular, it has been used to formalize and monitor DECLARE rules augmented with quantitative time constraints, and nonatomic activities and data-aware conditions (Montali et al. 2013a, b).

SeaFlows is a compliance checking framework that addresses design and runtime checking. It aims at encoding compliance states in an easily interpretable manner to provide advanced compliance diagnosis. The core concepts described in Ly (2013) and Ly et al. (2011) were implemented within the prototype named SeaFlows Toolset. With SeaFlows, compliance rules are modeled as *compliance rule graphs* (CRG).

The framework described in Santos et al. (2012) is based on the supervisory control theory. Through this approach, it is possible to supervise the execution of a process in a workflow management system by “blocking” those events that would lead to a violation of a given set of constraints. This can be considered as a very sophisticated form of proactive violation management, which is applicable only when the workflow system can be (at least partially) controlled by the monitor.

ECE rules (Bragaglia et al. 2011) are a domain-independent approach that was not specifically tailored for business process

monitoring. Two key features characterize ECE rules. First, they support an imperfect (i.e., fuzzy and probabilistic) matching between expected and occurred events and hence deal with several fine-grained degrees of compliance. Second, expected events can be decorated with countermeasures to be taken in case of a violation, hence providing first-class support for compensation mechanisms.

With BPPath (Sebahi 2012), Sebahi proposes an approach for querying execution traces based on XPath. BPPath implements a fragment of first-order hybrid logic and enables data-aware and resource-aware constraints.

An approach based on constraint programming is provided in Gomez-Lopez et al. (2013). The main goal of Gomez-Lopez et al. (2013) is to proactively detect compliance violations and to explain the root cause of the violation.

Giblin et al. (2006) presents an approach based on timed propositional temporal logic for transforming high-level regulatory constraints into REALM constraints that can be monitored during process runtime. The paper presents architectural considerations and an implementation of the transformation on the basis of a case study.

Hallé and Villemaire (2008) provides an approach based on LTL-FO⁺.LTL-FO⁺ is a linear temporal logic augmented with full first-order quantification over the data inside a trace of XML messages that focuses on monitoring data-aware constraints over business processes. The data is part of the messages that are exchanged between the process activities.

MONPOLY (Basin et al. 2008, 2011) is a runtime verification framework for security policies, specified in the logic MFOTL, a variant of LTL-FO with metric time. The high expressiveness of MFOTL allows one to express sophisticated compliance rules involving advanced temporal constraints, as well as data- and resource-related conditions.

In Narendra et al. (2008), the authors address the problem of continuous compliance monitoring, i.e., they evaluate the compliance of a process execution with respect to a set of policies at runtime. In particular, the policies are checked when some specific tasks are executed, which

are called control points. The authors define an optimization problem to find the right balance between the accuracy of the compliance checking and the number of control points (each control point has a verification cost).

Thullner et al. (2011) defines a framework for compliance monitoring able to detect violation and to suggest possible recovery actions after the violation has occurred. The framework is focused on the detection of different types of time constraints.

The work by Baresi et al. on Dynamo Baresi and Guinea (2005) constitutes one of the few techniques and tools dealing with monitoring (BPEL) web services against complex rules, going beyond the analysis of quantitative KPIs for service-level agreement. (Timed) Dynamo allows one to model ECA-like rules that mix information about the location (i.e., the target web services and operations), the involved data, and the temporal constraints. Dynamo provides continuous, reactive monitoring facilities based on the messages fetched so far, with sophisticated recovery and compensation mechanisms. Furthermore, the monitoring results can be aggregated and reported to the user, providing useful insights that go beyond a yes/no answer.

In Namiri and Stojanovic (2007), the authors describe an approach that is based on the patterns proposed by Dwyer et al. (1999). In particular, the authors introduce a set of control patterns. Control patterns are triggered by events (such as the execution of a controlled activity). When being triggered, conditions associated with the control are evaluated. In order to provide data for evaluating the associated conditions, the authors introduce a semantic mirror that is filled with runtime data of a process instance. For each control, actions to be carried out if a control fails may be specified.

Conclusion

In this chapter, we have presented the existing literature in the context of declarative process mining. From this analysis, we can derive that the works dealing with process discovery are mainly focused on DECLARE and, in general,

on a limited set of rule patterns. The works on conformance checking are based on larger languages covering the entire LTL or even richer languages like LDL. The approaches developed in the context of compliance monitoring are more variegated and are based on different declarative languages with different characteristics.

Cross-References

- [Automated Process Discovery](#)
- [Conformance Checking](#)

References

- Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P (2008) Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Trans Comput Log* 9(4):29:1–29:43
- Baresi L, Guinea S (2005) Dynamo: dynamic monitoring of WS-BPEL processes. In: CSOC, vol 3826, pp 478–483
- Basin DA, Klaedtke F, Müller S, Pfitzmann B (2008) Runtime monitoring of metric first-order temporal properties. In: FSTTCS, vol 2, pp 49–60
- Basin DA, Harvan M, Klaedtke F, Zalinescu E (2011) MONPOLY: monitoring usage-control policies. In: RV, vol 7186, pp 360–364
- Bellodi E, Riguzzi F, Lamma E (2010a) Probabilistic declarative process mining. In: KSEM, pp 292–303
- Bellodi E, Riguzzi F, Lamma E (2010b) Probabilistic logic-based process mining. In: CILC, pp 292–303
- Bernardi ML, Cimitile M, Di Francescomarino C, Maggi FM (2014a) Using discriminative rule mining to discover declarative process models with non-atomic activities. In: RuleML, pp 281–295
- Bernardi ML, Cimitile M, Maggi FM (2014b) Discovering cross-organizational business rules from the cloud. In: CIDM, pp 389–396
- Bernardi ML, Cimitile M, Di Francescomarino C, Maggi FM (2016) Do activity lifecycles affect the validity of a business rule in a business process? *Inf Syst* 62:42–59
- Borrego D, Barba I (2014) Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Syst Appl* 41(11): 5340–5352
- Bragaglia S, Chesani F, Mello P, Montali M, Sottara D (2011) Fuzzy conformance checking of observed behaviour with expectations. In: AI*IA 2011, vol 6934, pp 80–91
- Bragaglia S, Chesani F, Mello P, Montali M, Torroni P (2012) Reactive event calculus for monitoring global computing applications. In: Artikis A, Craven R, Kesim Çiçekli NK, Sadighi B, Stathis K (eds) Logic programs, norms and action, vol 7360. Springer, Heidelberg, pp 123–146
- Burattin A, Maggi FM, van der Aalst WMP, Sperduti A (2012) Techniques for a posteriori analysis of declarative processes. In: EDOC, pp 41–50
- Burattin A, Cimitile M, Maggi FM (2014) Lights, camera, action! Business process movies for online process discovery. In: BPM workshops, pp 408–419
- Burattin A, Cimitile M, Maggi FM, Sperduti A (2015) Online discovery of declarative process models from event streams. *IEEE Trans Serv Comput* 8(6):833–846
- Burattin A, Maggi FM, Sperduti A (2016) Conformance checking based on multi-perspective declarative process models. *Expert Syst Appl* 65:194–211
- Chesani F, Lamma E, Mello P, Montali M, Riguzzi F, Storari S (2009a) Exploiting inductive logic programming techniques for declarative process mining. *ToP-NoC* 2:278–295
- Chesani F, Mello P, Montali M, Riguzzi F, Sebastianis M, Storari S (2009b) Checking compliance of execution traces to business rules. In: BPM workshops, pp 134–145
- De Giacomo G, Vardi MY (2013) Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp 854–860
- De Giacomo G, De Masellis R, Grasso M, Maggi FM, Montali M (2014) Monitoring business metaconstraints based on LTL and LDL for finite traces. In: BPM, pp 1–17
- De Giacomo G, Maggi FM, Marrella A, Sardina S (2016) Computing trace alignment against declarative process models through planning. In: ICAPS, pp 367–375
- De Giacomo G, Maggi FM, Marrella A, Patrizi F (2017) On the disruptive effectiveness of automated planning for LTL f -based trace alignment. In: AAAI, pp 3555–3561
- De Masellis R, Maggi FM, Montali M (2014) Monitoring data-aware business constraints with finite state automata. In: ICSSP, pp 134–143
- Debois S, Hildebrandt TT, Laursen PH, Ulrik KR (2017) Declarative process mining for DCR graphs. In: SAC, pp 759–764
- Di Ciccio C, Mecella M (2012) Mining constraints for artful processes. In: BIS, pp 11–23
- Di Ciccio C, Mecella M (2013) A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM, pp 135–142
- Di Ciccio C, Mecella M (2015) On the discovery of declarative control flows for artful processes. *ACM Trans Manag Inf Syst* 5(4):24:1–24:37
- Di Ciccio C, Maggi FM, Mendling J (2014) Discovering target-branched declare constraints. In: BPM, pp 34–50
- Di Ciccio C, Maggi FM, Montali M, Mendling J (2015) Ensuring model consistency in declarative process discovery. In: BPM, Springer, pp 144–159. https://doi.org/10.1007/978-3-319-23063-4_9
- Di Ciccio C, Maggi FM, Mendling J (2016) Efficient discovery of target-branched declare constraints. *Inf Syst* 56:258–283

- Di Ciccio C, Maggi FM, Montali M, Mendling J (2017) Resolving inconsistencies and redundancies in declarative process models. *Inf Syst* 64:425–446
- Di Ciccio C, Maggi FM, Montali M, Mendling J (2018) On the relevance of a business constraint to an event log. *Inf Syst*
- Dwyer M, Avrunin G, Corbett J (1999) Patterns in property specifications for finite-state verification. In: ICSE, pp 411–420
- de Leoni M, Maggi FM, van der Aalst WMP (2012a) Aligning event logs and declarative process models for conformance checking. In: BPM, pp 82–97
- de Leoni M, Maggi FM, van der Aalst WMP (2012b) Aligning event logs and declarative process models for conformance checking. In: BPM, pp 82–97
- de Leoni M, Maggi FM, van der Aalst WM (2014) An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf Syst* 47:258–277
- Ferilli S (2014) Woman: logic-based workflow learning and management. *IEEE Trans Syst Man Cybern Syst* 44(6):744–756
- Giblin C, Müller S, Pfitzmann B (2006) From regulatory policies to event monitoring rules: towards model-driven compliance automation. Technical report. Report RZ 3662
- Gomez-Lopez M, Gasca R, Rinderle-Ma S (2013) Explaining the incorrect temporal events during business process monitoring by means of compliance rules and model-based diagnosis. In: EDOC workshops, pp 163–172
- Grando MA, van der Aalst WMP, Mans RS (2012) Reusing a declarative specification to check the conformance of different CIGs. In: BPM workshops, pp 188–199
- Grando MA, Schonenberg MH, van der Aalst WMP (2013) Semantic-based conformance checking of computer interpretable medical guidelines. In: BIOSTEC, vol 273, pp 285–300
- Haisjackl C, Zugal S, Soffer P, Hadar I, Reichert M, Pinggera J, Weber B (2013) Making sense of declarative process models: common strategies and typical pitfalls. In: BPMDS, pp 2–17
- Hallé S, Villemaire R (2008) Runtime monitoring of message-based workflows with data. In: ECOC, pp 63–72
- Kala T, Maggi FM, Di Ciccio C, Di Francescomarino C (2016) *A priori* and sequence analysis for discovering declarative process models. In: EDOC, pp 1–9
- Kowalski RA, Sergot MJ (1986) A logic-based calculus of events. *N Gener Comput* 4(1):67–95
- Lamma E, Mello P, Montali M, Riguzzi F, Storari S (2007a) Inducing declarative logic-based models from labeled traces. In: BPM, pp 344–359
- Lamma E, Mello P, Riguzzi F, Storari S (2007b) Applying inductive logic programming to process mining. In: ILP, pp 132–146
- Ly LT (2013) SeaFlows – a compliance checking framework for supporting the process lifecycle. PhD thesis, University of Ulm
- Ly LT, Rinderle-Ma S, Knuplesch D, Dadam P (2011) Monitoring business process compliance using compliance rule graphs. In: OTM conferences (1), pp 82–99
- Ly LT, Maggi FM, Montali M, Rinderle-Ma S, van der Aalst WMP (2013) A framework for the systematic comparison and evaluation of compliance monitoring approaches. In: EDOC, pp 7–16
- Ly LT, Maggi FM, Montali M, Rinderle-Ma S, van der Aalst WMP (2015) Compliance monitoring in business processes: functionalities, application, and tool-support. *Inf Syst* 54:209–234
- Maggi FM (2013) Declarative process mining with the declare component of ProM. In: BPM (Demos), vol 1021
- Maggi FM (2014) Discovering metric temporal business constraints from event logs. In: BIR, pp 261–275
- Maggi FM, Westergaard M (2014) Using timed automata for *a Priori* warnings and planning for timed declarative process models. *Int J Cooperative Inf Syst* 23(1)
- Maggi FM, Westergaard M (2017) Designing software for operational decision support through coloured Petri nets. *Enterprise IS* 11(5):576–596
- Maggi FM, Montali M, Westergaard M, van der Aalst WMP (2011a) Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: BPM 2011, vol 6896, pp 132–147
- Maggi FM, Mooij AJ, van der Aalst WMP (2011b) User-guided discovery of declarative process models. In: CIDM, pp 192–199
- Maggi FM, Westergaard M, Montali M, van der Aalst WMP (2011c) Runtime verification of LTL-based declarative process models. In: RV 2011, vol 7186, pp 131–146
- Maggi FM, Bose RPJC, van der Aalst WMP (2012a) Efficient discovery of understandable declarative process models from event logs. In: CAiSE, pp 270–285
- Maggi FM, Montali M, van der Aalst WMP (2012b) An operational decision support framework for monitoring business constraints. In: FASE, pp 146–162
- Maggi FM, Bose RPJC, van der Aalst WMP (2013a) A knowledge-based integrated approach for discovering and repairing Declare maps. In: CAiSE, pp 433–448
- Maggi FM, Burattin A, Cimitile M, Sperduti A (2013b) Online process discovery to detect concept drifts in LTL-based declarative process models. In: OTM, pp 94–111
- Maggi FM, Dumas M, García-Bañuelos L, Montali M (2013c) Discovering data-aware declarative process models from event logs. In: BPM, pp 81–96
- Maggi FM, Montali M, Di Ciccio C, Mendling J (2016) Semantical vacuity detection in declarative process mining. In: BPM, pp 158–175
- Maggi FM, Di Ciccio C, Di Francescomarino C, Kala T (2018) Parallel algorithms for the automated discovery of declarative process models. *Inf Syst.* <https://doi.org/10.1016/j.is.2017.12.002>

- Montali M, Pesic M, van der Aalst WMP, Chesani F, Mello P, Storari S (2010) Declarative specification and verification of service choreographies. *ACM Trans Web* 4(1):1–62
- Montali M, Chesani F, Mello P, Maggi FM (2013a) Towards data-aware constraints in Declare. In: SAC, pp 1391–1396
- Montali M, Maggi FM, Chesani F, Mello P, van der Aalst WMP (2013b) Monitoring business constraints with the event calculus. *ACM Trans Intell Syst Technol* 5(1):17:1–17:30
- Namiri K, Stojanovic N (2007) Pattern-based design and validation of business process compliance. In: CoopIS, On the move to meaningful internet systems, Springer, pp 59–76
- Narendra N, Varshney V, Nagar S, Vasa M, Bhamidipaty A (2008) Optimal control point selection for continuous business process compliance monitoring. In: IEEE SOLI, vol 2, pp 2536–2541
- Pesic M, Schonenberg H, van der Aalst WMP (2007) DECLARE: full support for loosely-structured processes. In: EDOC, pp 287–300
- Pichler P, Weber B, Zugal S, Pinggera J, Mendling J, Reijers HA (2011) Imperative versus declarative process modeling languages: an empirical investigation. In: BPM workshops, pp 383–394
- Pnueli A (1977) The temporal logic of programs. In: Annual IEEE symposium on foundations of computer science, pp 46–57
- Reijers HA, Slaats T, Stahl C (2013) Declarative modeling—an academic dream or the future for BPM? In: BPM, pp 307–322
- Rovani M, Maggi FM, de Leoni M, van der Aalst WMP (2015) Declarative process mining in healthcare. *Expert Syst Appl* 42(23):9236–9251
- Santos EAP, Francisco R, Vieira AD, FR Loures E, Busetti MA (2012) Modeling business rules for supervisory control of process-aware information systems. In: BPM workshops, vol 100, pp 447–458
- Schönig S, Di Cicco C, Maggi FM, Mendling J (2016a) Discovery of multi-perspective declarative process models. In: ICSOC, pp 87–103
- Schönig S, Rogge-Solti A, Cabanillas C, Jablonski S, Mendling J (2016b) Efficient and customisable declarative process mining with SQL. In: CAiSE, pp 290–305
- Sebahi S (2012) Business process compliance monitoring: a view-based approach. PhD thesis, LIRIS
- Silva NC, de Oliveira CAL, Albino FALA, Lima RMF (2014) Declarative versus imperative business process languages – a controlled experiment. In: ICEIS, pp 394–401
- Thullner R, Rozsnyai S, Schiefer J, Obweger H, Suntinger M (2011) Proactive business process compliance monitoring with event-based systems. In: EDOC workshops, pp 429–437
- van der Aalst WMP (2016) Process mining: data science in action. Springer, Berlin/Heidelberg
- van der Aalst WMP, de Beer HT, van Dongen BF (2005) Process mining and verification of properties: an approach based on temporal logic. In: CoopIS, pp 130–147
- van der Aalst WMP, Pesic M, Schonenberg H (2009) Declarative workflows: balancing between flexibility and support. *Comput Sci Res Dev* 23(2):99–113
- vanden Broucke SKLM, Vanthienen J, Baesens B (2014) Declarative process discovery with evolutionary computing. In: CEC, pp 2412–2419
- Westergaard M, Maggi FM (2011a) Declare: a tool suite for declarative workflow modeling and enactment. In: BPM (Demos)
- Westergaard M, Maggi FM (2011b) Modeling and verification of a protocol for operational support using coloured Petri nets. In: PETRI NETS, pp 169–188
- Westergaard M, Maggi FM (2012) Looking into the future: using timed automata to provide a priori advice about timed declarative process models. In: CoopIS, pp 250–267
- Zugal S, Pinggera J, Weber B (2011) The impact of testcases on the maintainability of declarative process models. In: BMMDS/EMMSAD, pp 163–177

Decomposed Process Discovery and Conformance Checking

Josep Carmona

Universitat Politècnica de Catalunya, Barcelona, Spain

Definitions

Decomposed process discovery and *decomposed conformance checking* are the corresponding variants of the two monolithic fundamental problems in process mining (van der Aalst 2011): *automated process discovery*, which considers the problem of discovering a process model from an event log (Leemans 2009), and *conformance checking*, which addresses the problem of analyzing the adequacy of a process model with respect to observed behavior (Munoz-Gama 2009), respectively.

The term *decomposed* in the two definitions is mainly describing the way the two problems are tackled operationally, to face their computational complexity by splitting the initial problem into smaller problems, that can be solved individually and often more efficiently.

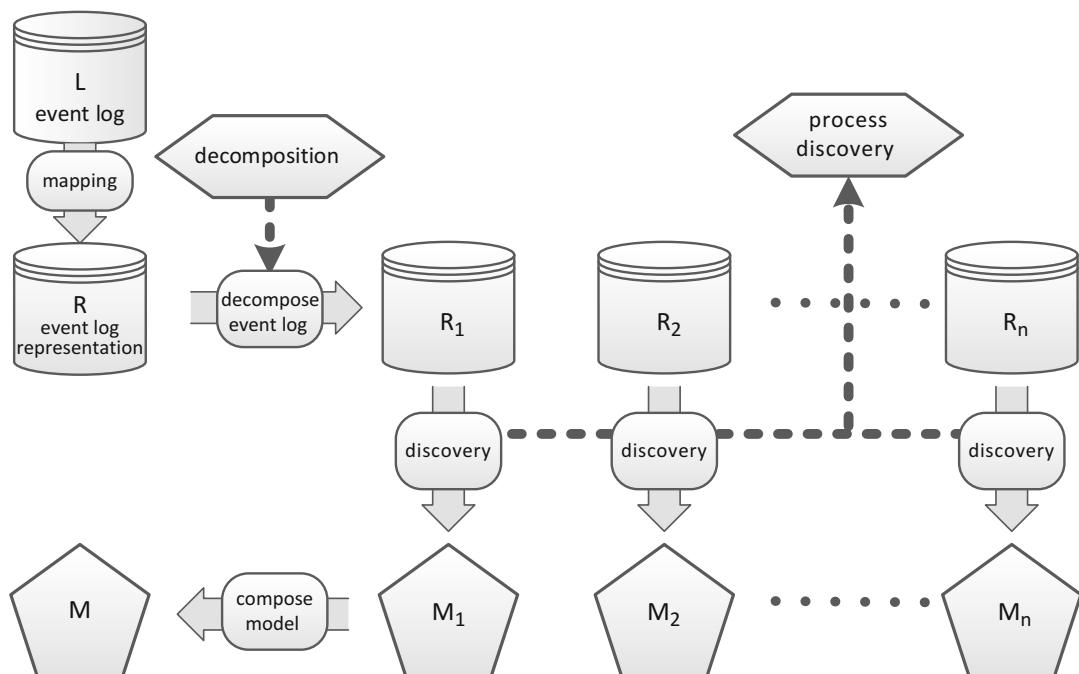
Overview

The input for process discovery is an *event log* (Mendling and Dumas 2009), from which a process model (typically a Petri net Murata 1989) needs to be produced. The input for conformance checking is an event log and a process model (again, typically a Petri net), from which a conformance artifact or a conformance diagnosis will be produced.

General View of Decomposed Process Discovery. The general view of decomposed process discovery is shown in Fig. 1. Initially, the log may be mapped to a different representation R , which is well suited to apply decomposition. Among the different alternatives, a common one is to do the decomposition on the log itself (so, $R = L$). An alternative is to derive a state-based representation, e.g., using the techniques from van der Aalst et al. (2010). Then, instead of directly deriving a process model from the representation of the event log (R), it is first partitioned, deriving

a set of representations R_1, \dots, R_n . In case the event log is used for decomposition, these will be *sublogs* L_1, \dots, L_n of L .

An important distinction should be done: the decomposition that is performed for decomposed process discovery and conformance checking is different from the decomposition that is done by *trace clustering* techniques. Trace clustering techniques partition the traces in the event log into multiple sets of traces, such that each trace in the original log can be found in one of the sublogs. In general, each sublog generated by trace clustering is a set of “similar” traces, and it corresponds to a “variant” of the process. For the purpose of decomposed process discovery, the decomposition does not consider dividing the traces in the log into multiple sets, but instead by splitting each trace into a set of *subtraces*. In other words, one trace is cut into multiple ones, denoted subtraces. After each trace has been split in this way, each set of subtraces is put together into one sublog. Intuitively, the idea is that each sublog produced in this way corresponds to a



Decomposed Process Discovery and Conformance Checking, Fig. 1 General view of decomposed process discovery. (Figure adapted from van der Aalst 2013)

stage or subprocess (or more generally a “fragment”) of the original process.

The available techniques to do the partition are enumerated below. Then, from each representation R_i , a process model M_i is obtained through the application of an automated process discovery technique. The output of decomposed process discovery can be the set of process models or fragments derived, or if some composition mechanism is disposable, a unique final model M that encompasses the unified behavior of the n derived process models.

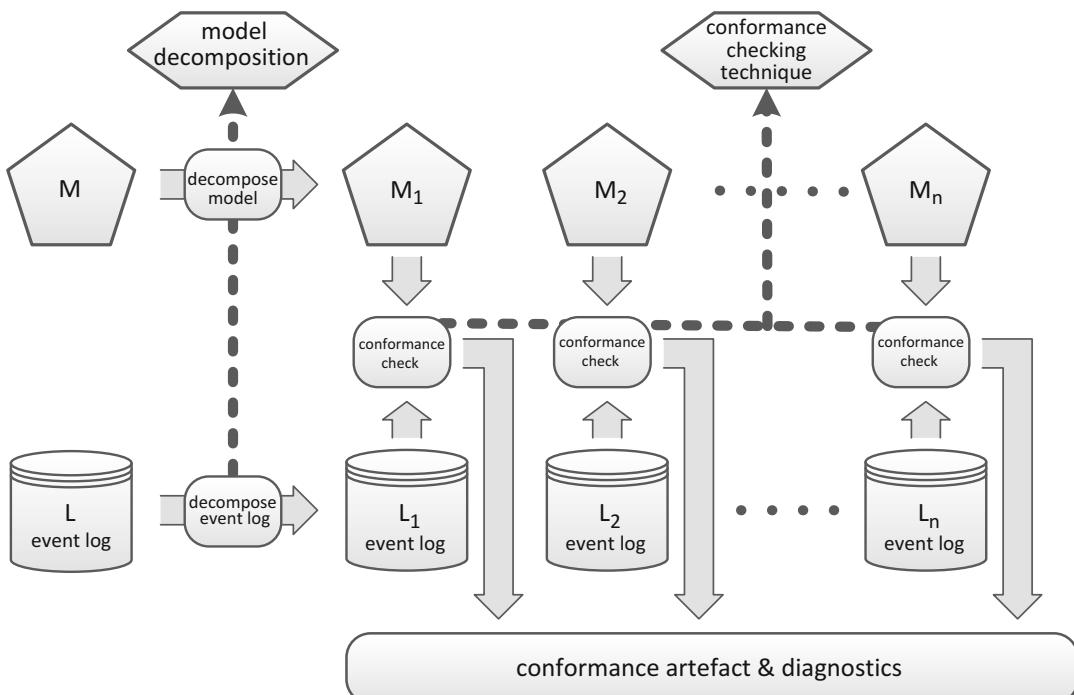
General View of Decomposed Conformance Checking

Checking. The general view of decomposed conformance checking is shown in Fig. 2. Although both process model and event log are decomposed, the process model decomposition is applied first, and then the event log decomposition is performed so that process model fragments and corresponding sublogs match on their alphabet of activities. Then, conformance checking techniques can be applied locally on

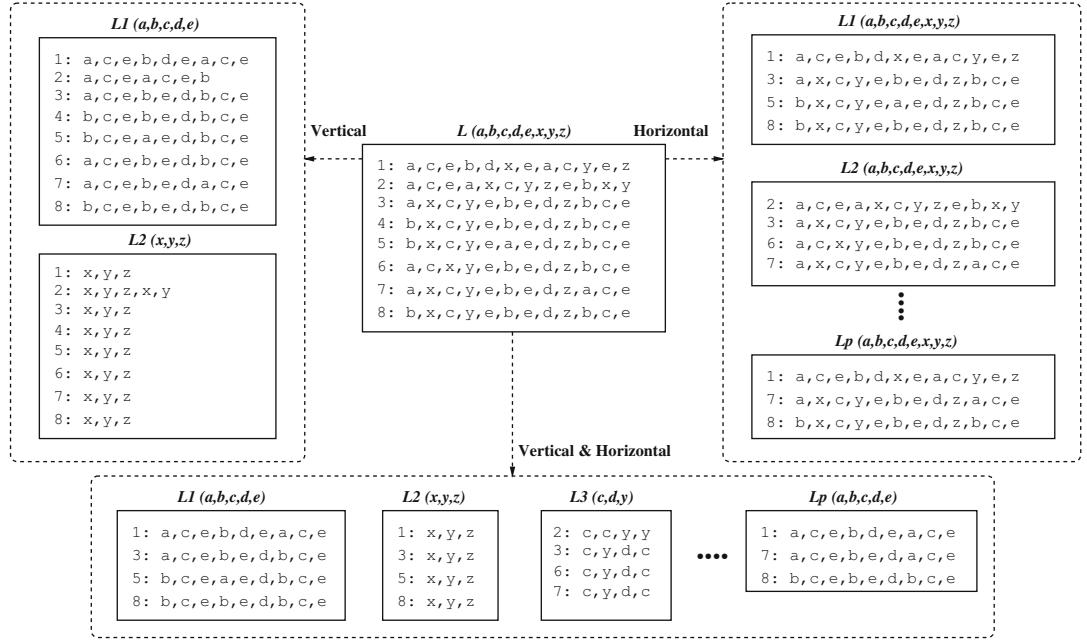
each pair of sublog and fragment generated. As conformance checking strongly relies on relating modeled and observed behavior (Muñoz-Gama 2009), the obtention of a conformance artifact like an *alignment* (Adriansyah 2014) between a sublog L_i and a fragment M_i can be materialized. Likewise, conformance diagnosis can already be obtained on these local problems.

The decompositions mentioned above can then be applied at different levels: *log decomposition* and *process model decomposition* (van der Aalst 2013). Below a general perspective on the two types of decomposition techniques is provided.

Log Decomposition. When the event log is not transformed to a different representation, decomposition can then be applied on top of the event log. Event logs can be either split horizontally, or projected vertically, or both. Figure 3 provides an example of such operations. The horizontal selection of traces can be done in many ways, but *trace clustering* (Greco et al. 2006; Ferreira



Decomposed Process Discovery and Conformance Checking, Fig. 2 General view of decomposed conformance checking. (Figure adapted from van der Aalst 2013)



Decomposed Process Discovery and Conformance Checking, Fig. 3 Log decomposition by either horizontal or vertical selections or a combination thereof

et al. 2007; Bose and van der Aalst 2009 b, c; Weerdt et al. 2013; Hompes et al. 2015) is usually applied. Notice that in the figure, sublogs arising from clusters can overlap, e.g., trace 3 belongs both to $L1$ and $L2$ in the horizontal selection of traces from L . Clearly, by decomposing a log into smaller sublogs, a linear factor on the complexity reduction can be accomplished.

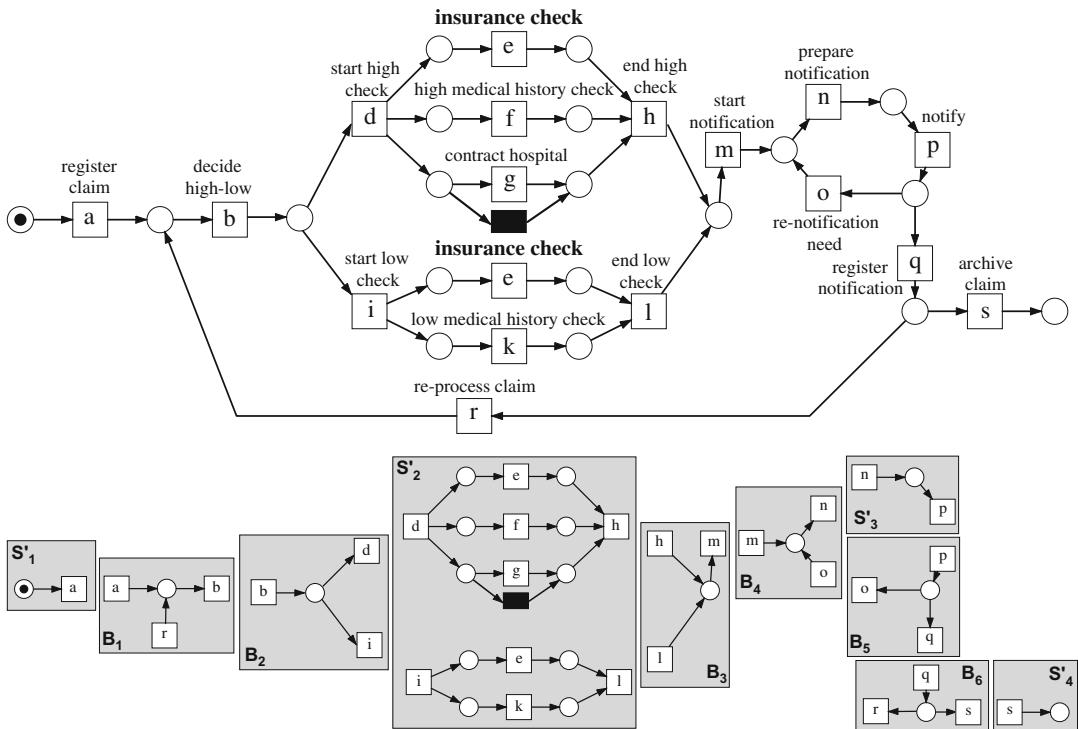
The vertical projection of log traces starts by selecting a proper subset \mathcal{A}' of the log's event alphabet \mathcal{A} . It then projects each trace σ in the log onto activities in \mathcal{A}' , i.e., removing from σ those events in $\mathcal{A} - \mathcal{A}'$, resulting in the trace $\sigma|_{\mathcal{A}'}$. The vertical projection can derive subsets $\mathcal{A}_1, \dots, \mathcal{A}_n$ with $\mathcal{A}_i \subset \mathcal{A}$, deriving the sublogs $L|_{\mathcal{A}_1}, \dots, L|_{\mathcal{A}_n}$. How to select those sets is the crucial decision; there are techniques based on the *directly-follows* relation from the α -algorithm (Carmona 2012) or hierarchical methods that learn patterns which become new labels in the alphabet (Bose and van der Aalst 2009a), among others. The vertical projection has in general a bigger impact in the complexity alleviation than the horizontal split, since the complexity of most of the techniques for discovery or confor-

mance checking is dominated by the size of the alphabet.

In case the log has been transformed into a different representation R , then other forms of decomposition are applicable. For instance, if the log is transformed into a *transition system* (Arnold 1994) which encompasses the behavior underlying in L , then decomposition techniques like the ones presented in de San Pedro and Cortadella (2016) can be applied.

Process Model Decomposition. Process model decomposition only makes sense for conformance checking, where apart from the event log, the process model is also an input. The intuitive idea is to break the process model into fragments and project vertically the log accordingly (see Fig. 2). This way, the problem instances tackled in decomposed conformance checking are significantly smaller than in the monolithic version.

Not every decomposition can be used in the scope of conformance checking: a *valid decomposition* (van der Aalst 2013) of a Petri net into fragments requires that the places and arcs



Decomposed Process Discovery and Conformance Checking, Fig. 4 Process model decomposition based on single-entry single-exit (SESE) components (Munoz-Gama et al. 2014)

of the net are partitioned among the fragments, meaning that the decomposition uniquely assigns each place to a single fragment. Transitions in the frontiers of the cuts deriving the fragments are replicated, while transitions inside one fragment cannot, i.e., transitions inside a fragment cannot occur in other fragments. Figure 4 shows an example of a valid decomposition of a process model. Examples of decompositions are the ones based on *passages* (van der Aalst 2012) or based on *single-entry single-exit* (SESE) (Munoz-Gama et al. 2014).

Key Research Findings

In general, the use of decomposition tends to make automated process discovery and conformance checking problems more tractable. However, current decomposition techniques cannot guarantee optimality in general or even to provide

the same outputs as the ones provided in the monolithic versions of the problems.

Key Research Findings in Decomposed Process Discovery.

Techniques for decomposed process discovery have appeared in different forms in the last years (Solé and Carmona 2012; Carmona 2012; van der Aalst 2013; van der Aalst and Verbeek 2014; van der Aalst et al. 2015; de San Pedro and Cortadella 2016). These approaches not only focus on alleviating the computation of the process discovery task but also consider other goals. For instance, a discovery approach based on van der Aalst (2013) was used in the framework that recently won the *Process Discovery Contest* (Carmona et al. 2016). Decomposition-based process discovery can also be applied to derive a different modeling notation, like it is presented in Solé and Carmona (2012) for *C-nets* (van der Aalst et al. 2011). It can also focus on deriving particular Petri net

subclasses, like *free choice*, *state machines*, or *marked graphs* (Carmona 2012; de San Pedro and Cortadella 2016).

Key Research Findings in Decomposed Conformance Checking. Techniques for decomposed conformance checking have been proposed in the last years (van der Aalst 2013; van der Aalst and Verbeek 2014; Munoz-Gama et al. 2014; vanden Broucke et al. 2014; de Leoni et al. 2014; Verbeek 2017). Apart from the aforementioned techniques for decomposing a process model through a valid decomposition (van der Aalst 2013; van der Aalst and Verbeek 2014; Munoz-Gama et al. 2014), other approaches that have a different focus have been presented. In vanden Broucke et al. (2014), the SESE-based decomposition approach is adapted to boost the computation so that it can be applied online. On a different perspective but again using SESE-based decomposition, the approach in de Leoni et al. (2014) shows how data can also be taken into account.

Fitness, i.e., the capability of a model in reproducing a trace, strongly influences the idea of valid decomposition: on a valid decomposition, the fitness of the whole model with respect to the trace can be regarded as the fitness of the individual fragments (see Fig. 2). Therefore, if none of the fragments has a fitness problem, then the initial model is fitting (van der Aalst 2013). The opposite is not true in general: a valid decomposition where some fragment contains a fitness problem does not necessarily imply that there exists a real fitness problem. Also, by separating the initial problem into pieces, important conformance artifact like *optimal alignments*, i.e., alignments between model and log which contain the minimal number of deviations, cannot be guaranteed. There has been recent work investigating the two aforementioned issues (Verbeek and van der Aalst 2016).

Examples of Application

All areas where process mining techniques have been applied are amenable for the application

of decomposed process discovery and conformance checking. It is suitable for large problem instances, e.g., in healthcare (Mans et al. 2015), software repositories, and finances, among many others.

D

Future Directions for Research

There exist several challenges to face, so that decomposed techniques can be applied in more situations. Below we provide some key challenges that may be faced in the next years:

- *Decomposition in the large*: the techniques available so far do not always guarantee a bound in the complexity of the operations performed. Fundamental research needs to be developed so that lightweight approaches are proposed that guarantee this bound, opening the door to real-time or online approaches.
- *Recomposing the results*: both in Figs. 1 and 2, one can see that the result of the decomposition may not always be a unique object, but several. To deploy a unique final result from these may be a challenge. For instance, in the case of decomposed conformance checking, to compute an alignment, it is well known that there are situations where only a best effort is possible (Verbeek and van der Aalst 2016). Therefore, research on how to recompose the results will be needed.
- *Decomposition beyond Petri nets*: most of the techniques available for decomposition rely strongly on the Petri net semantics. However, other modeling notations are also used, and in particular the industry adoption of Petri nets is low. Therefore, other notations like BPMN will be considered as input for decomposition techniques.
- *Decomposition beyond control flow*: very few approaches have been presented that consider not only the control flow of the process, but also other data attributes (de Leoni et al. 2014). However, the aforementioned techniques decompose only on the basis

of the control flow. Techniques that apply decomposition on different perspectives will enable a different perspective for decomposition.

- *Alternative theories for decomposed conformance checking:* currently, only the notion of valid decomposition has been explored as a viable strategy to decompose a process model. In the future, totally different alternatives to a valid decomposition may be studied.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Business Process Event Logs and Visualization](#)
- ▶ [Conformance Checking](#)
- ▶ [Trace Clustering](#)

References

- Adriansyah A (2014) Aligning observed and modeled behavior. PhD thesis, Technische Universiteit Eindhoven
- Arnold A (1994) Finite transition systems. Prentice Hall, Paris
- Bose RPJC, van der Aalst WMP (2009a) Abstractions in process mining: a taxonomy of patterns. In: Proceedings of the 7th international conference business process management, BPM 2009, Ulm, 8–10 Sept 2009, pp 159–175. https://doi.org/10.1007/978-3-642-03848-8_12
- Bose RPJC, van der Aalst WMP (2009b) Context aware trace clustering: towards improving process mining results. In: Proceedings of the SIAM international conference on data mining, SDM 2009, 30 Apr–May 2 2009, Sparks, pp 401–412
- Bose RPJC, van der Aalst WMP (2009c) Trace clustering based on conserved patterns: towards achieving better process models. In: BPM 2009 international workshops business process management workshops, Ulm, 7 Sept 2009. Revised papers, pp 170–181
- Carmona J (2012) Projection approaches to process mining using region-based techniques. Data Min Knowl Discov 24(1):218–246. <https://doi.org/10.1007/s10618-011-0226-x>
- Carmona J, de Leoni M, Depaire B, Jouck T (2016) Summary of the process discovery contest 2016. In: Proceedings of the BPM 2016 workshops. LNBIp, vol 281, pp 7–10
- de Leoni M, Munoz-Gama J, Carmona J, van der Aalst WMP (2014) Decomposing alignment-based conformance checking of data-aware process models. In: Proceedings on the move to meaningful internet systems: OTM 2014 conferences – confederated international conferences: CoopIS, and ODBASE 2014, Amantea, 27–31 Oct 2014, pp 3–20. https://doi.org/10.1007/978-3-662-45563-0_1
- de San Pedro J, Cortadella J (2016) Mining structured Petri nets for the visualization of process behavior. In: Proceedings of the 31st annual ACM symposium on applied computing, Pisa, 4–8 Apr 2016, pp 839–846. <http://doi.acm.org/10.1145/2851613.2851645>
- Ferreira DR, Zacarias M, Malheiros M, Ferreira P (2007) Approaching process mining with sequence clustering: experiments and findings. In: Proceedings of the business process management, 5th international conference, BPM 2007, Brisbane, 24–28 Sept 2007, pp 360–374
- Greco G, Guzzo A, Pontieri L, Saccà D (2006) Discovering expressive process models by clustering log traces. IEEE Trans Knowl Data Eng 18(8):1010–1027. <https://doi.org/10.1109/TKDE.2006.123>
- Hompes B, Buijs J, van der Aalst W, Dixit P, Buurman H (2015) Discovering deviating cases and process variants using trace clustering. In: Proceedings of the 27th benelux conference on artificial intelligence (BNAIC)
- Leemans S (2009) Automated process discovery. In: Sakr and Zomaya (2017), pp 288–289
- Mans R, van der Aalst WMP, Vanwersch RJB (2015) Process mining in healthcare – evaluating and exploiting operational healthcare processes. Springer briefs in business process management. Springer. <https://doi.org/10.1007/978-3-319-16071-9>
- Mendling J, Dumas M (2009) Business process event logs and visualization. In: Sakr and Zomaya (2017), pp 288–289
- Munoz-Gama J (2009) Conformance checking. In: Sakr and Zomaya (2017), pp 288–289
- Munoz-Gama J, Carmona J, van der Aalst WMP (2014) Single-entry single-exit decomposed conformance checking. Inf Syst 46:102–122. <https://doi.org/10.1016/j.is.2014.04.003>
- Murata T (1989) Petri nets: properties, analysis and applications. Proc IEEE 77(4):541–580
- Sakr S, Zomaya A (eds) (2017) Encyclopedia of big data technologies. Springer, Berlin-Heidelberg
- Solé M, Carmona J (2012) A high-level strategy for c-net discovery. In: 12th international conference on application of concurrency to system design, ACSD 2012, Hamburg, 27–29 June 2012, pp 102–111
- van der Aalst WMP (2011) Process mining – discovery, conformance and enhancement of business processes. Springer, Berlin-Heidelberg
- van der Aalst WMP (2012) Decomposing process mining problems using passages. In: Application and theory of petri nets – 33rd international conference, PETRI NETS 2012, Hamburg, 25–29 June 2012. Proceedings, pp 72–91. https://doi.org/10.1007/978-3-642-31131-4_5
- van der Aalst WMP (2013) Decomposing petri nets for process mining: a generic approach. Distrib Parallel Databases 31(4):471–507. <https://doi.org/10.1007/s10619-013-7127-5>

- van der Aalst WMP, Verbeek HMW (2014) Process discovery and conformance checking using passages. *Fundam Inform* 131(1):103–138
- van der Aalst WMP, Rubin VA, Verbeek HMW, van Dongen BF, Kindler E, Günther CW (2010) Process mining: a two-step approach to balance between underfitting and overfitting. *Softw Syst Model* 9(1):87–111. <https://doi.org/10.1007/s10270-008-0106-z>
- van der Aalst WMP, Adriansyah A, van Dongen BF (2011) Causal nets: a modeling language tailored towards process discovery. In: Proceedings of the CONCUR 2011 – concurrency theory – 22nd international conference, CONCUR 2011, Aachen, 6–9 Sept 2011, pp 28–42
- van der Aalst WMP, Kalenkova AA, Rubin VA, Verbeek E (2015) Process discovery using localized events. In: Proceedings of the application and theory of Petri Nets and concurrency – 36th international conference, PETRI NETS 2015, Brussels, 21–26 June 2015, pp 287–308
- vanden Broucke SKLM, Munoz-Gama J, Carmona J, Baesens B, Vanthienen J (2014) Event-based real-time decomposed conformance analysis. In: On the move to meaningful internet systems: OTM 2014 conferences – confederated international conferences: CoopIS, and ODBASE 2014, Amantea, 27–31 Oct 2014, Proceedings, pp 345–363. https://doi.org/10.1007/978-3-662-45563-0_20
- Verbeek HMW (2017) Decomposed replay using hiding and reduction as abstraction. *Trans Petri Nets Other Models Concurr* 12:166–186
- Verbeek HMW, van der Aalst WMP (2016) Merging alignments for decomposed replay. In: Proceedings of the application and theory of petri nets and concurrency – 37th international conference, PETRI NETS 2016, Toruń, 19–24 June 2016, pp 219–239
- Weerdt JD, vanden Broucke SKLM, Vanthienen J, Baesens B (2013) Active trace clustering for improved process discovery. *IEEE Trans Knowl Data Eng* 25(12):2708–2720. <https://doi.org/10.1109/TKDE.2013.64>

Deep Analytics Benchmark

- ▶ End-to-End Benchmark

Deep Learning

- ▶ Flood Detection Using Social Media Big Data Streams

Deep Learning on Big Data

Gianluigi Folino¹, Massimo Guarascio¹, and Maryam A. Haeri²

¹ICAR-CNR (Institute for High Performance Computing and Networks, National Research Council), Rende, Italy

²Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

D

Definitions

Deep Learning is a family of machine learning methods based on the composition of multiple (simple) processing layers, in order to learn complex (nonlinear) functions, which can be used to cope with many challenging application scenarios (e.g., *Computer Vision, Natural Language Processing, Speech Recognition and Genomics* (Le Cun et al. 2015)).

Overview

In the last few years, advances in digital sensors, computation, communications, and storage technologies and the large diffusion of IoT devices facilitated the production of huge collections of heterogeneous data, which are also susceptible to rapid changes over time. The term *Big Data* is used to define this kind of phenomenon (Wu et al. 2014).

Deep learning (DL) includes a wide range of machine learning techniques that aims at training artificial neural networks (ANNs) composed of a large number of hidden layers. These methods have been effectively used to tackle different types of problems (e.g., *Computer Vision, Natural Language Processing, Speech Recognition, and Recommender Systems*) and allow to handle large amounts of data in an efficient way. Specifically, a deep neural network (DNN) works in a hierarchical way; indeed, each subsequent layer of the architecture generates a set of features with a higher level of abstraction compared with

the previous one. Therefore, it is possible to extract informative and discriminative features even from raw data (e.g., pixels of images, biological sequences, or user preference data). This capability is one of the most important and disruptive advances introduced by the deep learning paradigm; indeed, usually no feature engineering phase and domain experts are required to select a good set of features.

Different from traditional algorithms, based on shallow hierarchies that fail to explore and learn highly complex patterns, Deep Learning-based techniques allow to intrinsically exploit the large amounts of data featuring big data (*Volume*). Moreover, as Deep Learning-based approaches allow to extract data abstractions and representations at different levels, they are also suitable to analyze raw data provided in different formats and by different types of source (*Variety*). Finally, deep architectures can be used within infrastructures for big data storage and analysis (e.g., *Hadoop* and *Spark*) and can exploit GPUs to parallelize the computation and to reduce the learning times. Hadoop, in particular, is widely used in industry for the development of big data applications such as *spam filter*, *click stream analysis*, *social recommendation*, etc. That allows to handle the speed at which new data are being produced in these big data scenarios (*Velocity*).

Deep learning can be used to tackle different types of learning task. The most common machine learning task for DNNs is *supervised learning*. In this case, DNNs allow to learn discriminative patterns for classification purposes, when labeled data are available in direct or indirect forms. The most known DNN architectures for supervised learning are *Convolutional Neural Networks* and *Recurrent Neural Networks*. Deep neural networks for unsupervised or generative learning are used when no information about class labels is available and allow to extract high-order correlation from data for pattern analysis or summarization purposes. Several architectures have been developed to address these learning tasks; in particular *Autoencoders*, (*Deep-/Restricted*) *Boltzmann Machine*, and *Deep Belief Network* are largely recognized as effective tech-

niques. Finally, semi-supervised learning permits to handle the case where only a small subset of the training data is labeled (e.g., they use unlabeled data to define the cluster boundaries and use a small amount of labeled data to label the clusters). To this aim, *Reinforcement learning* can be used to train the weights so that, given the state of the current environment, DNNs can find which action the agent should take next, in order to maximize expected rewards.

In this chapter, first we introduce some basic concepts required to understand how DNNs work. Then, we focus on the most diffused supervised and unsupervised architectures exploited in literature for handling big data. Finally, we discuss some interesting open issues and future research directions.

Deep Neural Networks

ANNs are computational models that try to replicate the neural structure of the brain. In a nutshell, the brain learns and stores information as patterns. These patterns can be highly complex and allow us to perform many tasks, such as recognizing individual from the image of their faces also considering different angles of view. This process of storing information as patterns, applying those patterns and making inferences, represents the approach behind ANNs.

The artificial neuron is the main building block of a neural network and encompasses some interesting capabilities. Its behavior can be thought as composed of two steps. First, the linear combination of the inputs is computed (a weight, usually limited in the range 0–1, is associated with each value of the input vector). Note that the summation function often takes an extra input value b having a weight of 1, which represents the threshold or the bias of a neuron.

The sum-of-product value is provided as input for the second step. Here, the *activation function* is executed to compute the final output the neuron. The activation function limits the output in the range [0, 1] or alternately [-1, 1].

An artificial neural network is essentially a combination of neurons stacked in layers.

The most simple structure for an ANN includes two levels: the neurons placed in the first

layer allow to handle the inputs by distributing them to the next layer, while only the second layer performs a computation. Each of the inputs $x_1, x_2, x_3, \dots, x_N$ are connected to every artificial neuron in the output layer through the connection weight. Since every value of outputs $y_1, y_2, y_3, \dots, y_N$ is calculated from the same set of input values, each output is varied based on the connection weights.

The above-described architecture is quite simple and does not provide high-level computational capabilities. However, this structure can be easily enriched by stacking further computation layers in the network. In this case, the input nodes transfer the data to the processing elements in the first hidden layer, and then the outputs from the first hidden layer are forwarded to the next one and so on. This iterative process is called *forward propagation*, and this type of architecture is an example of *feedforward neural network*, since the connectivity graph does not have any directed loops or cycles. Moreover, if each neuron of a layer is connected with each neuron of the next layer, the neural network is called *fully-connected*.

DNNs exacerbate this scheme by exploiting hundreds of levels in order to learn complex functions, obtained by combining nonlinear operations. The depth of the DNN is the number of levels/layers contained in the architecture.

A DNN can be fully specified by four basic components:

- The *topology of the network*, i.e., the number of layers and the connections among the neurons;
- The learning algorithm (variants of the *gradient descent*);
- The *activation functions* to use within the nodes;
- The *loss function* to minimize.

This chapter introduced the basic concepts that characterize DNNs and their main principles; for the interested reader, more details are supplied in the following reference paper (Le Cun et al. 2015).

Key Research Findings

As mentioned in the introduction, DL gathers a wide range of machine learning methods and techniques, including several layers of nonlinear processing, stacked into a hierarchical scheme. On the basis of their learning goals (e.g., detection/prediction or synthesis/generation), the architectures and the techniques proposed in this area can be broadly classified into architectures for supervised or unsupervised tasks.

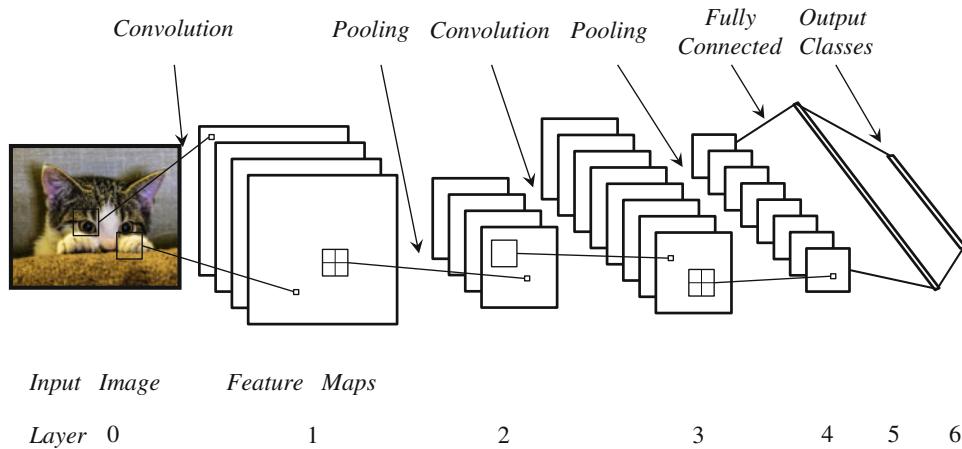
In unsupervised learning, task-specific supervision information (e.g., target class labels) is not used in the learning process. Many DNN architectures belonging to this category have been used for feature extraction and for cleaning data; typical examples are RBMs, DBNs, DBMs, and generalized denoising autoencoders. In supervised environments, the concept to discover is available and provided as target in the learning phase. Typical supervised architectures are CNNs, RNNs, and LSTMs.

On the following, the most common architectures for supervised and unsupervised learning are detailed, with a particular attention to the analysis of big data.

Convolutional Neural Networks

In big data context, CNNs are typically used to tackle *Face Detection* and *Image Recognition* tasks. Several architectures have been proposed to cope with the massive amount of data produced by different types of big data providers (e.g., *Security Cameras and Video Surveillance Systems*, *Autonomous cars*, etc.). Specifically, the deep architectures proposed in Szegedy et al. (2015) allow to analyze massive datasets of images by exploiting the potentialities of deep CNNs.

A convolutional neural network is composed of a stacking of *convolutional layers*. Each layer produces a higher-level abstraction of the input data, called a *feature map* (fmap), which stores essential information about the data. Usually, CNN architectures assume to work with two-dimensional data (e.g., images), and this makes the forward function more efficient to implement and permits to largely reduce the amount of parameters in the network. A basic CNN



Deep Learning on Big Data, Fig. 1 Example of a CNN architecture

architecture can be designed as a sequence of layers, where each layer transforms one volume of activations to another, by using a differentiable function. In literature, three main types of layer are used to learn CNNs: *Convolutional Layer*, *Pooling Layer*, and *Fully-Connected Layer*. In Fig. 1, a simple CNN architecture that combines these layers is shown.

On the one hand, convolutional layers allow to discover local conjunctions of features from the previous layer; on the other hand pooling layers combine semantically similar features in a single feature. It is common to insert a pooling layer in-between successive convolutional layers. In fact, it permits to reduce the dimension of the representation and the amount of parameters in the network and consequently control also the overfitting.

Recurrent Neural Networks and Long Short Time Memory Networks

Many typical DNN architectures have no memory, and the output for a given input is always the same independently from the sequence of inputs previously given to the network. On the contrary RNNs, of which LSTMs are a well-known variant, keep an internal memory to make long-term dependencies influence the output.

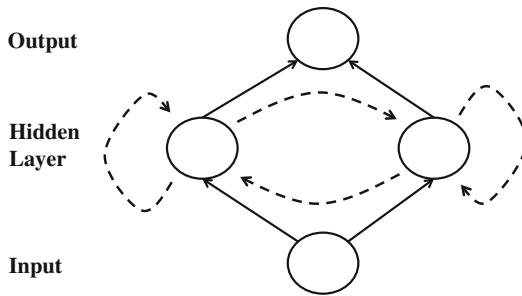
In a nutshell, in these networks, some intermediate operations generate values that are stored internally and used as inputs for other operations.

RNNs receive as input two values, representing the current state and the recent past of the input, which are combined to determine how they respond to new data.

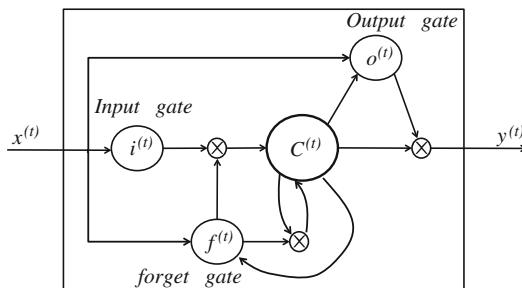
These networks assume that the sequence of the input states is an important information; therefore, the added memory allows to carry out tasks that feedforward networks cannot perform. In particular, this sequential information is kept in the hidden state of the RNNs. Basically, RNNs can be defined as feedforward neural networks featured by edges that join subsequent time steps, including the concept of time within the model. Specifically, like feedforward neural networks, these learning architectures do not exhibit cycles among the standard edges; however, those that connect adjacent time steps (named *recurrent edges*) can have cycles, including self-connections.

At time t , the nodes having recurrent edges receive the input value $x^{(t)}$ from the current data point and also the value $h^{(t-1)}$ from the hidden node, which keeps the previous state of the network. The output $\tilde{y}^{(t)}$ at each time t is computed by exploiting as input the hidden node values $h^{(t)}$ at time t . Input $x^{(t-1)}$ at time $t - 1$ can influence $\tilde{y}^{(t)}$ at time t and later by means of the recurrent connections. Figure 2 shows the above-described scheme.

LSTM improves RNN by introducing the concept of memory cell, a computation unit that



Deep Learning on Big Data, Fig. 2 Simple RNN architecture



Deep Learning on Big Data, Fig. 3 Memory cell structure

replaces traditional nodes in the hidden layer of a network. By using these cells, these networks are able to overcome issues related to the training of the RNN. Specifically, a LSTM can be thought as an evolution of a standard RNN with a hidden layer, but each standard neuron in the hidden layer is replaced by a memory cell, shown in Fig. 3. Each memory cell contains a node with a self-connected recurrent edge of fixed weight equal to one, ensuring that the gradient can pass across many time steps without vanishing.

The term *long short-term memory* is due to the following idea. RNNs keep long-term memory as weights, which can change slowly during the learning phase and encodes information about the data. LSTM introduces an intermediate storage step by means of memory cells in order to overcome this problem. Memory cells are computation units composed of standard nodes connected according to a specific pattern, where the main novelty element is the presence of multiplicative nodes (labeled in Fig. 3 with the symbol \times). *Gates* are the main components of a LSTM ar-

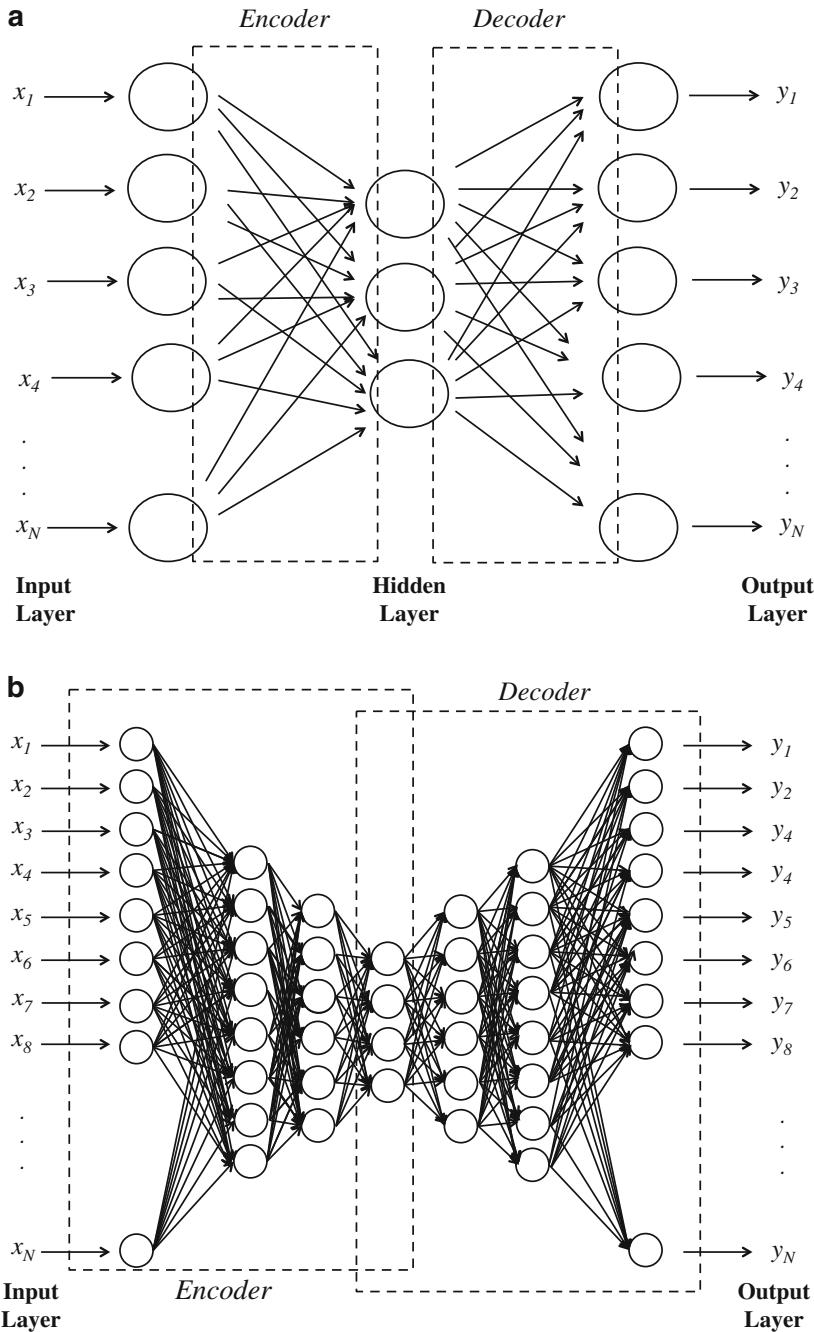
chitecture. They are sigmoidal units that are activated by the current data point $x^{(t)}$ and by the hidden layer at the previous time step. The main component of the memory cell is the node labeled as $C^{(t)}$ that contains a linear activation function. It has a self-connected recurrent edge having a fixed unitary weight.

Recurrent neural networks have been widely used in many different big data scenarios (e.g., *image/video captioning*, *word prediction*, *translation*, *image processing*). The most common application is within the framework of natural language processing (NLP). Recently applications such as handwriting recognition (Graves et al. 2007), language translation (Sutskever et al. 2014), and image and video annotation (Byeon et al. 2015) take advantage of RNNs to incorporate temporal information and to improve the quality of prediction.

Autoencoders

In literature, different types of generative deep architectures have been proposed, but energy-based models and deep autoencoders are the most widely used (Bengio 2009). Autoencoder, named also autoassociator, is a particular (feedforward) NN architecture, whereby the target of the network is the data itself. It is mainly used for *dimensionality reduction* or for *learning efficient encoding*. Specifically, *Machine Translation*, *Word Embeddings and Phrase Representation*, *Document Clustering*, and *Sentiment Analysis* are some challenging (big data) environments where deep architectures have been successfully used (Chandar et al. 2014; Lebret and Collobert 2015; Hinton and Salakhutdinov 2006; Glorot et al. 2011).

Basically, an autoencoder can be thought as an unsupervised feature extraction method. Its network topology is similar to a multilayer ANN. The most simple architecture (shown in Fig. 4a) includes three components: (i) an input layer of raw data to be encoded (e.g., images or audio signals); (ii) a (stack of) hidden layer(s), characterized by a considerable smaller number of neurons; and (iii) an output layer having the same size (i.e., the same number of neurons) of the input layer. An autoencoder is called deep, if the



Deep Learning on Big Data, Fig. 4 Autoencoder architecture. (a) Simple autoencoder. (b) Deep autoencoder

number of its hidden layers is greater than one (Fig. 4b).

Specifically, an autoencoder can be conceptually thought as composed of two subnets, respectively, named *encoder* and *decoder*. The

former allows to learn a mapping between the input layer and the hidden layers (*encoding*) and the second a mapping between the features extracted by the encoder and the output layer (*decoding*). The encoder receives as

input a set of features and extracts a smaller set of new (high-level) features (by using the sigmoid as activation function for each neuron). Hence, the decoder converts these features back to the original input attributes. An interesting variant of this architecture is called (*Stacked*) *Denoising Autoencoder*, which shares some properties with restricted Boltzmann machines and can be used for generative purposes (Vincent et al. 2008). In a nutshell, it tries to minimize the reconstructing error due to applied stochastic noise to the input. In this way, it can be used as a pre-training strategy for learning DNNs.

Boltzmann Machines (BMs)

An alternative approach to autoencoders assumes that the observed data are the result of a stochastic process, which can be interpreted in neural network terms. Deep Boltzmann machine is a very popular deep architecture with generative capability. Basically, it is a parametrized generative model representing a probability distribution modeled by means of a neural network with two (or more) layers. First layer is used to handle the input (i.e., the component of observations x). Hidden units allow to model dependencies between the components of the observations.

These architectures are designed with several stacked layers of hidden variables, and no connection is established among the neurons belonging to the same layer. In practice, it is a particular form of *Boltzmann Machine* (see Fig. 5a), an ANN whose structure is featured by symmetric and bidirectional edges connecting neurons belonging to the same layer. A DBM with a single hidden layer is called *Restricted Boltzmann Machine* (RBM) (see Fig. 5b). It is represented as a bipartite graph, where all the visible neurons are connected to all the hidden ones, and there are no connections among the neurons belonging to the same layer.

DBMs have the potential of learning internal representations that become increasingly complex, and they are able to solve object and speech recognition problems. Moreover, DBMs are often extended with an output layer: the weights

learned can be used as initial weights for the supervised training of the resulting ANN.

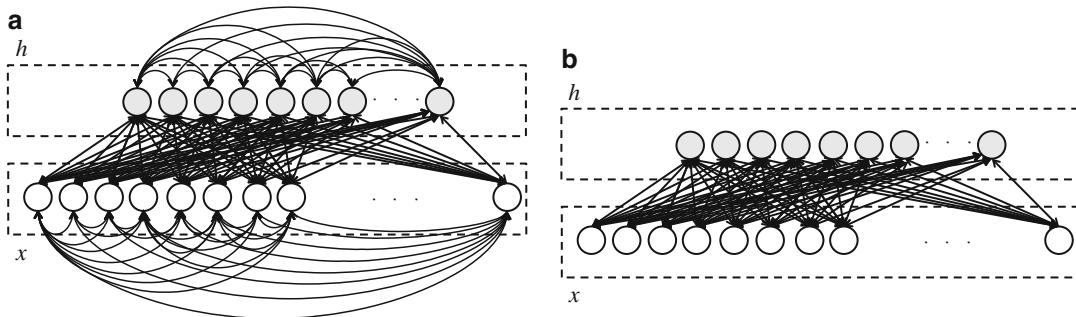
Deep Boltzmann Machines (DBMs) are widely used to analyze large volume of data in different contexts as *Handwritten Digits Recognition*, *Collaborative Filtering* (Salakhutdinov et al. 2007), *Phone Recognition* (Dahl et al. 2010), *Transportation Network Congestion Monitoring and Prediction* (Ma et al. 2015) and *Facial Landmark Tracking* (Wu et al. 2013).

D

Examples of Application

Deep learning can provide layered and hierarchical architectures for representing and learning complex data. Moreover, deep learning can be applied to both labeled/supervised and unlabeled/unsupervised tasks concerning big data (Najafabadi et al. 2015). These features make it a valuable method for analyzing big data and extracting salient patterns from complex and massive data. In the last few years, a number of applications concerning deep learning for big data have been developed; in the following, we cite some of the most significant ones, which includes semantic indexing, data tagging, remote sensing, and fault detection.

Semantic Indexing: Information retrieval is one of the most important tasks when working with big data. Every day, high volumes of data are generated from different sources such as social networks, web pages, and marketing systems. The information retrieval systems need to manage these data. Semantic indexing is a preferable method to present the data rather than store it in a raw format. Semantic indexing provides a data representation, which is more appropriate for faster search and knowledge extraction. The data representations generated by deep learning are more abstract and high level in comparison with other traditional techniques. Examples of these applications include different data types such as text (Huang et al. 2013), image (Zhang et al. 2013), and video data. For example, Shen et al. (2014) suggested a series of new latent semantic models based on convolutional neural networks



Deep Learning on Big Data, Fig. 5 Boltzmann machine architectures. (a) Boltzmann machine. (b) Restricted Boltzmann machine

to extract low-dimensional semantic vectors for search engine queries of web documents.

Discriminative Tasks: Deep learning can be used for discriminative tasks. It extracts nonlinear features from data; subsequently, linear models can be used to find a discriminative function. This property is really useful for applications such as object recognition and image comprehension in big data. For example, in Jia et al. (2016), the authors used different types of deep neural networks for image recognition problems such as gender recognition and face recognition.

Transfer Learning and Domain Adaptation:

In machine learning, transfer learning or inductive transfer is defined as a process, which utilizes the knowledge obtained during solving (learning) one problem, and applied to another different but related problem (Torrey and Shavlik 2009). The properties of big data provide a good opportunity to apply transfer learning and domain adaptation. Deep learning is a good tool to identify similar factors for different data-driven forms and for different problem domains (Bengio 2012). In Shin et al. (2016), the authors evaluated transfer learning by using convolutional neural networks trained over ImageNet on two different diagnosis applications: thoracoabdominal lymph node detection and interstitial lung disease classification.

Anomaly and Fraud Detection: Detecting anomalies and frauds is a challenging problem, especially in the case of big data. In applications such as electronic payments, network flows, and

social networks, frauds and anomalies should be detected coping with a large amount of data. Deep learning can be considered as an appropriate machine learning tool to detect anomalies and frauds, especially in the case of high dimensionality. For example, the authors of Paula et al. (2016) proposed an unsupervised deep learning method to detect frauds in exports. Moreover, Erfani et al. (2016) suggested using deep belief networks to extract robust features from high-dimensional data and then using support vector machine (SVM) to build a one-class model to detect anomalies.

Traffic Flow Prediction: Predicting the traffic flow in accurate and timely fashion is very important for intelligent transportation systems, and deep learning is appropriate to this task. As an example in Lv et al. (2015), the authors used a stacked autoencoder to learn traffic flow features and trained it with a greedy layer-wise method.

The applications of deep learning in big data are not limited to these topics; just to cite a few examples, remote sensing (Kussul et al. 2017) and fault detection (Ren et al. 2017) are two interesting fields in which these techniques were successfully applied for big data problems.

Future Directions for Research

Big data and deep learning are strictly connected as interesting research themes. Indeed, big data issues can benefit from the application of Deep Learning, and in addition, some hard problems

and applications can be solved by deep learning techniques by using big data developments and tools.

Some of the most relevant issues to be addressed by using deep learning techniques exploiting the potentialities of big data techniques include:

- **Large scale Deep Learning:** using parallel and distributed infrastructures permits to train in acceptable times really large deep networks or to train in near real-time deep networks. However, many issues must be addressed to make the process efficient, such as the model of parallelism, the efficient partitioning of the networks and of the layers, reducing the overhead of the messages, etc.
- **Real time data streaming:** only the usage of big data resources can address problematics as data changes and streaming in real-time; in addition, issues as the integration of multiple data streams, often coming from heterogeneous IoT objects, the need of adapting the layers or the parameters of the networks to frequent changes in data makes it necessary the adoption of big data solutions.
- **Heuristics to automatically generate or customize deep networks:** tuning the structure and the parameters of deep neural networks requires extensive and expensive training phases; big data technologies, by employing distributed heuristic and meta-heuristic algorithms for supporting this phase can strongly speedup this process.

On the other side, deep learning networks can be really helpful to cope with many modern big data problems and applications. We would like to mention, among the others, the following applications: natural language processing, human vision, video tagging, hybrid deep learning, self-driving cars, and cybersecurity problems.

Finally, an open field of research concerns the integration of different deep learning strategies, such as convolutional neural networks, deep belief network and Boltzmann machines, which can be efficiently used to cope with unstructured data

and to deal with the entire automatic process of analyzing these data, from the analysis and the extraction of information from audio and video files to the recognition of objects and faces and to the transcription of audio tracks and up to recognize real-time fake news.

D

References

- Bengio Y (2009) Learning deep architectures for AI. *Found Trends Mach Learn* 2(1):1–127
- Bengio Y (2012) Deep learning of representations for unsupervised and transfer learning. In: Proceedings of ICML workshop on unsupervised and transfer learning, pp 17–36
- Byeon W, Breuel TM, Raue F, Liwicki M (2015) Scene labeling with LSTM recurrent neural networks. In: CVPR, IEEE Computer Society, pp 3547–3555
- Chandar APS, Lauly S, Larochelle H, Khapra MM, Ravindran B, Raykar V, Saha A (2014) An autoencoder approach to learning bilingual word representations. In: Proceedings of the 27th international conference on neural information processing systems – vol 2, NIPS’14. MIT Press, Cambridge, MA, pp 1853–1861, <http://dl.acm.org/citation.cfm?id=2969033.2969034>
- Dahl G, Ranzato M, rahman Mohamed A, Hinton GE (2010) Phone recognition with the mean-covariance restricted Boltzmann machine. In: Lafferty JD, Williams CKI, Shawe-Taylor J, Zemel RS, Culotta A (eds) Advances in neural information processing systems 23. Curran Associates, Inc., pp 469–477
- Erfani SM, Rajasegarar S, Karunasekera S, Leckie C (2016) High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recogn* 58:121–134
- Glorot X, Bordes A, Bengio Y (2011) Domain adaptation for large-scale sentiment classification: a deep learning approach. In: Proceedings of the twenty-eight international conference on machine learning, ICML
- Graves A, Fernández S, Liwicki M, Bunke H, Schmidhuber J (2007) Unconstrained online handwriting recognition with recurrent neural networks. In: Proceedings of the 20th international conference on neural information processing systems, NIPS’07. Curran Associates Inc., pp 577–584
- Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507. <https://doi.org/10.1126/science.1127647>, <http://www.sciencemag.org/cgi/content/abstract/313/5786/504>
- Huang PS, He X, Gao J, Deng L, Acero A, Heck L (2013) Learning deep structured semantic models for web search using clickthrough data. In: Proceedings of the 22nd ACM international conference on conference on information & knowledge management. ACM, pp 2333–2338

- Jia S, Lansdall-Welfare T, Cristianini N (2016) Gender classification by deep learning on millions of weakly labelled images. In: 2016 IEEE 16th international conference on data mining workshops (ICDMW). IEEE, pp 462–467
- Kussul N, Lavreniuk M, Skakun S, Shelestov A (2017) Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geosci Remote Sens Lett* 14(5):778–782
- Le Cun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444
- Lebret R, Collobert R (2015) “The sum of its parts”: joint learning of word and phrase representations with autoencoders. *CoRR abs/1506.05703*
- Lv Y, Duan Y, Kang W, Li Z, Wang FY (2015) Traffic flow prediction with big data: a deep learning approach. *IEEE Trans Intell Transp Syst* 16(2):865–873
- Ma X, Yu H, Wang Y, Wang Y (2015) Large-scale transportation network congestion evolution prediction using deep learning theory. <https://doi.org/10.1371/journal.pone.0119044>
- Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E (2015) Deep learning applications and challenges in big data analytics. *J Big Data* 2(1):1
- Paula EL, Ladeira M, Carvalho RN, Marzagão T (2016) Deep learning anomaly detection as support fraud investigation in Brazilian exports and anti-money laundering. In: 2016 15th IEEE international conference on machine learning and applications (ICMLA). IEEE, pp 954–960
- Ren H, Chai Y, Qu J, Ye X, Tang Q (2017) A novel adaptive fault detection methodology for complex system using deep belief networks and multiple models: a case study on cryogenic propellant loading system. *Neurocomputing* 275:2111–2125
- Salakhutdinov R, Mnih A, Hinton G (2007) Restricted Boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on machine learning, ICML’07. ACM, New York, pp 791–798. <https://doi.org/10.1145/1273496.1273596>, <http://doi.acm.org/10.1145/1273496.1273596>
- Shen Y, He X, Gao J, Deng L, Mesnil G (2014) Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the 23rd international conference on world wide web. ACM, pp 373–374
- Shin HC, Roth HR, Gao M, Lu L, Xu Z, Nogues I, Yao J, Mollura D, Summers RM (2016) Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE Trans Med Imaging* 35(5):1285–1298
- Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Proceedings of the 27th international conference on neural information processing systems – vol 2, NIPS’14. MIT Press, Cambridge, MA, pp 3104–3112. <http://dl.acm.org/citation.cfm?id=2969033.2969173>
- Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Computer vision and pattern recognition (CVPR)
- Torrey L, Shavlik J (2009) Transfer learning. In: Soria E, Martin J, Magdalena R, Martinez M, Serrano A (eds) Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, vol 1. IGI Global, Hershey, p 242
- Vincent P, Larochelle H, Bengio Y, Manzagol P (2008) Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on machine learning, ICML’08, pp 1096–1103
- Wu Y, Wang Z, Ji Q (2013) Facial feature tracking under varying facial expressions and face poses based on restricted Boltzmann machines. In: 2013 IEEE conference on computer vision and pattern recognition, Portland, 23–28 June 2013, pp 3452–3459. <https://doi.org/10.1109/CVPR.2013.443>
- Wu X, Zhu X, Wu GQ, Ding W (2014) Data mining with big data. *IEEE Trans Knowl Data Eng* 26(1):97–107
- Zhang S, Yang M, Wang X, Lin Y, Tian Q (2013) Semantic-aware co-indexing for image retrieval. In: Proceedings of the IEEE international conference on computer vision, pp 1673–1680

Definition of Data Streams

Alessandro Margara¹ and Tilmann Rabl²

¹Politecnico di Milano, Milano, Italy

²Database Systems and Information

Management Group, Technische Universität Berlin, Berlin, Germany

Synonyms

[Event streams](#); [Information flows](#)

Definitions

A data stream is a countably infinite sequence of elements. Different models of data streams exist that take different approaches with respect to the mutability of the stream and to the structure of stream elements. Stream processing refers to analyzing data streams on-the-fly to produce new results as new input data becomes available. Time is a central concept in stream processing: in almost all models of streams, each stream ele-

ment is associated with one or more timestamps from a given time domain that might indicate, for instance, when the element was generated, the validity of its content, or when it became available for processing.

Overview

A data stream is a countably infinite sequence of elements and is used to represent data elements that are made available over time. Examples are readings from sensors in an environmental monitoring application, stock quotes in financial applications, or network data in computer monitoring applications.

A stream-based application analyzes elements from streams as they become available to timely produce new results and enable fast reactions if needed (Babcock et al. 2002; Cugola and Margara 2012).

In the last decade, several technologies have emerged to address the processing of high-volume, real-time data without requiring custom code (Stonebraker et al. 2005). They are commonly referred to as stream processing systems, and they are the topic of this section on “Big Stream Processing.”

In the remainder of this entry, we overview the main models of data streams and stream processing systems, and we briefly introduce some central concepts in stream processing, namely, time and windows. Furthermore, we summarize the main requirements of stream processing.

Stream Models

Streams can be structured or unstructured. In a structured stream, elements follow a certain format or schema that allows for additional modeling of the stream. In contrast, unstructured streams can have arbitrary contents often resulting from combining streams from many sources (these are also referred to as event showers or event clouds Doblander et al. 2014).

There are three major models for structured streams, which differentiate in how the elements of the stream are related to and influence each other: the turnstile model, the cash register

model, and the time series model. The most general model is the *turnstile model*. In this model, the stream is modeled as a vector of elements, and each element s_i in the stream is an update (increment or decrement) to an element of the underlying vector. The size of the vector in this model is the domain of the stream elements. This model is also the model typically used in traditional database systems, where there are inserts, deletes, and updates to the database. In the *cash register model*, elements in the stream are only additions to the underlying vector, but elements can never leave the vector again. This is similar to databases recording the history of relations. Finally, the *time series model* treats every element in the stream s_i as a new independent vector entry. As a result, the underlying model is a constantly increasing vector and generally unbounded vector. Because each element can be processed individually in this model, it is frequently used in current stream processing engines.

Time

Time is a central concept in many stream processing applications, either because they are concerned with updating their view of the world by taking into account *recent* data received from streams or because they aim to detect temporal trends in the input streams.

For this reason, in most models of stream and stream processing, data elements are associated with some timestamp from a given time domain. Common semantics of time include *event time*, which is the time when the element was produced, and *processing time*, which is the time when the stream processing system starts processing the element (Akidau et al. 2015).

Different time semantics introduce different problems in terms of order and synchronization. Intuitively, while event time and processing time should ideally be equal, in practice unsynchronized clocks at the producers as well as variable communication and processing delays produce a skew between event time and processing time which is not only non-zero but also highly variable (Akidau 2015). On the one hand, processing elements in event time order is necessary in

many application scenarios. On the other hand, this requires waiting for out-of-order elements and reorder them before processing. This is a nontrivial problem, which is analyzed in detail in the entry related to “Time management.”

Windows

Windows are one of the core building blocks of virtually all stream processing systems. They define bounded portions of elements over an unbounded stream. They are used to perform computations that would be impossible (nonterminating) in the case of unbounded data, such as computing the average value of all the elements in a stream of numbers.

The most common types of windows are *count-based* and *time-based* windows. The former define their size in terms of the number of elements that they include, while the latter define their size in terms of a time frame and include all the elements with a timestamp included in that time frame.

In both cases, we distinguish between *sliding* windows, which continuously advance with the arrival of new elements, thus always capturing new elements, and *tumbling* windows, which can accumulate multiple elements before moving (Botan et al. 2010).

More recently, new types of windows have been defined to better capture the needs of applications. They include *session* and *data-defined* windows that are variable in size and define their boundaries based on the data elements: for instance, in a software monitoring application, a window can include all and only the elements that refer to a session opened by a specific user of that software (Akidau et al. 2015).

Stream Processing

Several types of stream processing systems exist that are specialized on different types of computations (Cugola and Margara 2012).

Data stream management systems are designed to update the answers of *continuous queries* as new data becomes available. They typically adopt declarative abstractions similar to traditional database query languages that they enrich with constructs such as windows to deal

with the unbounded nature of the input data (Arasu et al. 2006).

Complex event processing systems, instead, aim to detect (temporal) patterns over the input stream of elements (Etzion and Niblett 2010; Luckham 2001). These systems are discussed in the entry on “Event recognition.”

Modern Big Data stream processing systems, such as Flink (Carbone et al. 2015), provide general operators to transform input streams into output streams, thus offering the possibility to implement heterogeneous streaming applications by integrating these operators. These systems are designed to scale to multiple processing cores and computing nodes and persist intermediate results for fault tolerance. Flink and other systems are discussed in several entries within this section of the encyclopedia. The “Introduction to stream processing” entry overviews the main classes of algorithms for stream processing.

General Requirements of Stream Processing

Data streams could theoretically be stored and processed with traditional database systems or other analytic solutions. However, real-time processing of streams introduces specific requirements that stream processing systems need to satisfy (Stonebraker et al. 2005). Although the following requirements are not necessarily fulfilled by all current stream processing systems, they are frequently necessary in stream processing applications.

In order to do advanced analytics on streams, which involve more than a single operation, a stream processing system needs to integrate (pipeline) basic operations with each other. This can be done by (micro-)batching the stream, that is, splitting it in small chunks such that each operator processes a chunk at a time, or by true streaming, where each element is processed individually.

Because streams are often generated in a distributed fashion, for example, in sensor networks, a streaming system must handle small inconsistencies in the stream, such as out-of-order data, missing values, or delayed values. This is related to the topic of time management, discussed

earlier. Even if a stream contains small inconsistencies, the result of the stream processing needs to be predictable. This demands for clear semantics of the operations available and for robust mechanisms that guarantee deterministic results.

A high-level interface to specify stream processing jobs is desirable. The section on stream processing languages and abstractions gives an overview of current solutions.

An area of active research is the combination of streaming and batch data. Interestingly, this combination forms the missing link between database systems and streaming systems. Early attempts to address this challenge have led to the idea of a lambda architecture, which consists of separate systems for streaming and batch processing (Marz and Warren 2015). However, especially problems with robustness and model mismatches call for an integrated solution. Current streaming systems address this by enabling advanced state management (Carbone et al. 2017; Affetti et al. 2017).

Future Directions of Research

Streaming is a very active area of research. However, there are several topics that have found limited attention in current research but will be increasingly important for future applications and systems. Here, we will highlight two areas that found little attention so far.

Current stream processing systems frequently differentiate themselves from database systems in that they see all streams and all analytic jobs as conceptually unbounded. However, in many scenarios, streams as well as the analytic jobs on them are bounded. Furthermore, both can arrive at high frequencies, which is a challenge for current systems.

Another challenge, not solved by current systems, is transactional guarantees. Current stream processing systems provide basic guarantees like processing each stream element exactly once or at least once but have no concept of transactions that span multiple stream elements or operations (Affetti et al. 2017).

Summary

In this entry, we presented an overview on stream processing and introduced the basic concepts and terminology adopted in other entries of this section.

D

Cross-References

- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Stream Processing Languages and Abstractions](#)
- ▶ [Management of Time](#)

References

- Affetti L, Margara A, Cugola G (2017) Flowdb: integrating stream processing and consistent state management. In: Proceedings of the international conference on distributed and event-based systems, DEBS'17. ACM, pp 134–145. <https://doi.org/10.1145/3093742.3093929>
- Akida T (2015) The world beyond batch: streaming 101
- Akida T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E, Whittle S (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. VLDB 8(12):1792–1803. <https://doi.org/10.14778/2824032.2824076>
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. VLDB 15(2):121–142. <https://doi.org/10.1007/s00778-004-0147-z>
- Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the symposium on principles of database systems, PODS'02. ACM, pp 1–16. <https://doi.org/10.1145/543613.543615>
- Botan I, Derakhshan R, Dindar N, Haas L, Miller RJ, Tatbul N (2010) Secret: a model for analysis of the execution semantics of stream processing systems. VLDB 3(1–2):232–243. <https://doi.org/10.14778/1920841.1920874>
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink: stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Comm Data Eng 36(4):28–38.
- Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in Apache flink®: consistent stateful distributed stream processing. Proc VLDB 10(12):1718–1729. <https://doi.org/10.14778/3137765.3137777>

- Cugola G, Margara A (2012) Processing flows of information: from data stream to complex event processing. *ACM Comput Surv* 44(3):15:1–15:62. <https://doi.org/10.1145/2187671.2187677>
- Doblander C, Rabl T, Jacobsen HA (2014) Processing big events with showers and streams. In: Rabl T, Poess M, Baru C, Jacobsen HA (eds) Specifying big data benchmarks. Springer, Berlin/Heidelberg, pp 60–71
- Etzion O, Niblett P (2010) Event processing in action. Manning Publications, Greenwich
- Luckham DC (2001) The power of events: an introduction to complex event processing in distributed enterprise systems. Addison-Wesley, Boston
- Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning Publications, Greenwich
- Stonebraker M, Çetintemel U, Zdonik S (2005) The 8 requirements of real-time stream processing. *SIGMOD Rec* 34(4):42–47. <https://doi.org/10.1145/1107499.1107504>

Degrees of Separation and Diameter in Large Graphs

Pierluigi Crescenzi¹ and Andrea Marino²

¹Dipartimento di Matematica e Informatica,
Università di Firenze, Florence, Italy

²Dipartimento di Informatica, Università di Pisa,
Pisa, Italy

Synonyms

Approximated neighborhood function; Computing average distance; Distance distribution; Maximum eccentricity in real-world networks

Definitions

Given a (di)graph $G = (V, E)$ (strongly) connected, where $n = |V|$ and $m = |E|$ (note that, since the graph is connected, we have $m \geq n - 1$), the distance between two vertices $u, v \in V$ is the number of edges along the shortest path from u to v and is denoted as $d(u, v)$. The number of nodes separating u and v , i.e., $d(u, v) - 1$, is also called degree of separation.

The eccentricity of a node u is $\text{ecc}(u) = \max_{v \in V} d(u, v)$, which measures in how many hops u can reach any other node in the graph. Hence, the diameter D (resp. the radius R) of G is defined as the maximum (resp. minimum) eccentricity among all the nodes, i.e., $D = \max_{u \in V} \text{ecc}(u)$ (resp. $R = \min_{u \in V} \text{ecc}(u)$).

Given a node $u \in V$ and an $h \in [D]$ (where, for any positive integer x , $[x]$ denotes the set $\{1, 2, \dots, x\}$), we call $B_h(u)$ the set $\{v : v \in V, d(u, v) \leq h\}$ and, similarly, $N_h(u)$ the set $\{v : v \in V, d(u, v) = h\}$, which are, respectively, the sets of nodes reachable from u with at most (resp. exactly) h edges. We denote $N_1(u)$ also as $N(u)$.

Moreover, we call B_h (resp. N_h) the fraction of pairs of nodes $x, y \in V$ such that $d(x, y) \leq h$ (resp. $d(x, y) = h$). Clearly, for $h > 0$, we have $N_h = B_h - B_{h-1}$, and the average distance of G is defined as $\sum_{u, v \in V} \frac{d(u, v)}{n^2} = \sum_h h \cdot N_h$.

Overview

The problem with computing the average distance and the diameter in the case of huge networks (like, for instance, the Facebook friendship network and the Internet Movie Database collaboration network) is the time and space complexity. Indeed, this basically consists of computing all-pairs shortest paths. If the number of nodes is very large (millions or billions), the time complexity of this operation is not acceptable. Moreover, the space required to store huge networks and the intermediate data used by the algorithms into the main memory of the computer might be prohibitive. For this reason, a combination of graph compression techniques (Boldi and Vigna 2003) and of algorithmic methods has been proposed and practically used in order to compute both the average and the maximum distance within a graph. In this entry we describe some algorithmic techniques for computing the distance distribution, which can be roughly divided into two main categories: classical and sketch-based sampling. Moreover, we describe very simple algorithm for computing the diameter

of huge real-world complex networks, which is in practice extremely efficient.

Lower bounds and techniques. As distance distribution, radius, and diameter can be computed using any algorithm computing all-pairs shortest path (in short, APSP), their total time cost can be $O(nm)$, for instance, computing n breadth-first searches (in short, BFSs), one for each node. This cost is clearly prohibitive in real-world huge networks, but it has been proved that, unfortunately, this worst-case time bound seems hard to improve unless the so-called strong exponential time hypothesis (SETH) is false (Impagliazzo et al. 2001). Indeed, under this hypothesis, even deciding whether a graph has diameter 2 or 3 requires $\Omega(n^2)$ (Roditty and Williams 2013).

For this reason, heuristics or approximation algorithms are often used. In the case of distance distribution, approximation algorithms have been designed to approximate N_h with bounded absolute error (Crescenzi et al. 2011) or to approximate B_h with bounded relative error (Cohen 1994, 2015; Boldi et al. 2011; Flajolet and Martin 1985). These approximations turn out to be approximations also for the average distance (Boldi et al. 2011; Backstrom et al. 2012).

Approximation algorithms have been also proposed in the case of diameter computation (Roditty and Williams 2013; Abboud et al. 2016; Cairo et al. 2016). However, by the reduction provided by Roditty and Williams (2013), we have that unless SETH fails, $\Omega(n^2)$ time is required to get a $(3/2 - \epsilon)$ -approximation algorithm for computing the diameter even in the case of sparse graphs.

In the case of the radius and diameter of real-world networks, some heuristics have been shown to work surprisingly well (Magnien et al. 2009; Borassi et al. 2015; Akiba et al. 2015). Some of them are designed to approximate the diameter with a constant number of BFSs providing lower (or upper) bounds. Even if their approximation ratio can be asymptotically 2 (Crescenzi et al. 2013), they often turn out to be tight in practice (Magnien et al. 2009; Crescenzi et al. 2010). Other heuristics have been designed to compute

exactly the diameter. Even though, consistently with the time lower bound provided by Roditty and Williams (2013), these algorithms may be $O(nm)$, they turn out to be linear in practice in real-world graphs.

In the following we will discuss the approximation methods for the distance distribution and then we will focus on the exact computation of diameter.

D

Key Research Findings

On Approximating the Distance Distribution

Classical Sampling

The idea of classical sampling is to select a random set of vertices U and to approximate the distribution of distances of the whole graph with the distribution of distances starting from U (Crescenzi et al. 2011). In other words, given a random set $U \subseteq V$, for any h , with $h \in [D]$, classical sampling approximates N_h with $N_h(U)$, where $N_h(U) = |\{(u, v) \in U \times V : d(u, v) = h\}|/(|U|n)$. Classical sampling is often classified as a Monte Carlo method and, in the field of network analysis, has been applied, for example, for closeness centrality (Eppstein and Wang 2001) and betweenness centrality (Brandes and Pich 2007).

The algorithm works as follows: (i) perform a random sample of k vertices from V obtaining the multiset $U = \{u_1, u_2, \dots, u_k\} \subseteq V$; (ii) run iteration $i = 1, 2, \dots, k$, computing the distances $d(u_i, v)$ for all $v \in V$, by executing a BFS traversal of G starting from vertex u_i ; and (iii) return the approximation $N_h(U)$. The running time of the algorithm is $O(km)$, with additional space of $O(n)$. By applying the Azuma-Hoeffding bound (Hoeffding 1963), it is possible to prove that choosing $k = \Theta(\epsilon^{-2} \log n)$, this algorithm computes an approximation of the distance distribution N_h of G whose absolute error is bounded by ϵ , with high probability.

As the number of connected pairs is not known a priori in directed weakly connected graphs, this

method does not directly extend to this kind of graph.

Sketch-Based Sampling

These methods aim to study the cumulative distance distribution B_h by noting that for any u :

$$\begin{aligned} B_1(u) &= \{u\} \cup N(u) \\ B_h(u) &= B_{h-1}(u) \cup \bigcup_{v \in N(u)} B_{h-1}(v). \end{aligned}$$

This relationship immediately gives an exact dynamic programming algorithm that in D steps computes the sets $B_h(u)$ for any u and B_h .

The time cost is $\Theta(Dm\Delta)$, where Δ is the maximum degree in G , and the memory usage is $\Theta(n^2)$, since the size of $B_{h-1}(u)$ for any u can be $\Theta(n)$. Sketch-based approaches represent the sets $B_h(u)$ for each u , in a compressed approximated form of (almost) constant size k (typically $O(\log(n))$). By using sketches in place of real sets, this allows to obtain a dynamic programming approximation algorithm that costs $O(Dmk)$ time and $\Theta(nk)$ space, which is shown in Algorithm 1 and explained next using sketches. Note that the number of iterations needed by the dynamic programming algorithm is the diameter of G , which can be computed by using the methods shown in this entry. Whenever this is not possible, one can stop the algorithm when the amount of sketch changes are below a certain threshold (Palmer et al. 2002; Boldi et al. 2011).

Sketches. Given a set A , a sketch $S(A)$ is a compressed form of representation of A of size $O(k)$, with $k \in \mathbb{N}$, providing the following operations:

INIT ($S(A)$) How a sketch $S(A)$ for A is initialized.

UPDATE ($S(A), u$) How a sketch $S(A)$ for A modifies when an element u is added to A .

UNION ($S(A), S(B)$) Given two sketches for A and B , provide a sketch for $A \cup B$.

SIZE ($S(A)$) Estimate the number of distinct elements of A .

The following two requirements are needed for sketches: (i) Given two sketches $S(A)$ and

Algorithm 1: Approximation algorithm for distance distribution

```

Input : Graph  $G = (V, E)$ ,  $D$  the diameter for  $G$  or an upper bound.
Output: An estimate for  $B_h$ , with  $h \in [D]$ .
1 for  $u \in V$  do
2   INIT ( $S(B_1(u))$ )
3   for  $v \in N(u)$  do
4     UPDATE ( $S(B_1(u)), v$ )
5 Output  $\sum_u \text{SIZE}(S(B_1(u))) / n^2$ 
6 for  $h \in 2 \dots D$  do
7   for  $u \in V$  do
8      $S(B_h(u)) \leftarrow S(B_{h-1}(u))$ 
9     for  $v \in N(u)$  do
10       $S(B_h(u)) \leftarrow$ 
11        UNION( $S(B_h(u)), S(B_{h-1}(v))$ )
11 Output  $\sum_u \text{SIZE}(S(B_h(u))) / n^2$ 

```

$S(B)$ for any two sets A and B , $S(A \cup B)$ can be computed just by looking at $S(A)$ and $S(B)$; (ii) The order in which the elements are added and adding any element twice does not affect the sketch. We assume that $|A| > k > 1$.

In the following, we present the most popular methods for sketches that have been used to compute distance distributions.

k -min. A k -min sketch includes the item of smallest rank in each of k independent permutations (Cohen 1997). Hence, a sketch $S(A)$ is a sequence of exactly k entries, a_1, \dots, a_k , where each entry can be an element of A or \perp .

Similar ideas have been applied for near duplicate document detection in Broder (1997).

Let r_1, r_2, \dots, r_k be mapping functions $r_i : U \rightarrow \{1/n, 2/n, 3/n, \dots, 1\}$, setting $r(\perp) = \infty$.

INIT ($S(A)$): $a_i = \perp$ for any i .

UPDATE ($S(A), u$): for every $i \in [k]$, if $r_i(a_i) > r_i(u)$, a_i is replaced with u .

UNION ($S(A), S(B)$): return $\{c_1, \dots, c_k\}$ such that $c_i = \arg\min_{x \in \{a_i, b_i\}} \{r_i(x)\}$.

SIZE ($S(A)$): return $\frac{k-1}{\sum_{a_i \in S(A)} -\ln(1-r_i(a_i))}$

Note that, as $|A| > 1$, $r_i(a_i) \neq 1$ for each $a_i \in S(A)$. The mean relative error is bounded by $0.79/\sqrt{(k-2)}$ (Cohen 1994, 1997).

Bottom- k . This paradigm relies on the technique introduced by Cohen and Kaplan (2008, 2007a,b). Given a mapping $r : U \rightarrow \{1/n, 2/n, \dots, 1\}$ and a subset A of U , we denote as $H_k(A)$ the first k elements of A according to r and with $\max(A)$ the element of A having maximum rank. A bottom- k sketch for A is $H_k(A)$.

INIT ($S(A)$): $S(A) = \emptyset$.

UPDATE ($S(A), u$): If $|S(A)| < k$, add u to $S(A)$. Otherwise, if $r(\max(S(A))) > r(u)$, replace $\max(S(A))$ with u .

UNION ($S(A), S(B)$): return $H_k(S(A) \cup S(B))$

SIZE ($S(A)$): return $\frac{k-1}{r(\max(S(A)))}$

It corresponds to the sampling without replacement version of the k -min approach.

LogLog counters. This approach exploits the LogLog counters introduced in Flajolet and Martin (1985). The implementation of this paradigm for computing distance distribution is called ANF (Palmer et al. 2002). Here a sketch $S(A)$ is a sequence of exactly k entries, a_1, \dots, a_k , where each entry is a sequence of h bits. Given k partition functions $p_i : U \rightarrow \{1, 2, \dots, h\}$, for each j , with $j \in [h]$, the bit $a_{i,j} = 1$ if there exists an element in A that is mapped to j according to p_i , 0 otherwise. Each partition function p_i is such that $1/2$ of the elements are randomly mapped to 1, $1/4$ of the elements are mapped to 2, ..., $1/2^i$ of them are mapped to i .

INIT ($S(A)$): $a_{i,j} = 0$ for any i, j .

UPDATE ($S(A), u$): for any i , let $p_i(u) = j$, set $a_{i,j}$ to 1.

UNION ($S(A), S(B)$): return $\{c_1, \dots, c_k\}$ where c_i is the OR between a_i and b_i .

SIZE ($S(A)$): let b be the average position of the least zero bits in a_1, \dots, a_k (i.e., $b = \frac{1}{k} \sum_{i=1}^k \min\{j - 1 : a_{ij} = 0\}$), return $2^b / 0.77351$.

The standard error is bounded by $0.78/\sqrt{k}$ using space $\log n + O(1)$ bits.

HyperLogLog Counters. This approach is inspired by the method proposed in Flajolet et al.

(2007). We refer to the description given by Boldi et al. (2011). In this case, let h be a hash function from U to 2^∞ . As in the case of LogLog counters, a sketch $S(A)$ is a sequence of exactly k entries a_1, \dots, a_k where $k = 2^b$ and b is given in input. Each entry a_i is a counter. Let $h_b(x)$ be the sequence of the b leftmost bits of $h(x)$, while let $h^b(x)$ be the remaining bits. Moreover, given a binary sequence w , let $\rho^+(w)$ be the number of leading zeroes in w plus one.

Intuitively, as in the case of LogLog counters, the elements of U are partitioned in k groups according to their value of $h_b(x)$, and in the sketch $S(A)$ in a_i , there is the maximum $\rho^+(h^b(y))$ among the elements y of A mapped in the i -th group.

INIT ($S(A)$): $a_i = -\infty$ for any i .

UPDATE ($S(A), u$): Let $i = h_b(u)$. Set a_i equal to $\max\{a_i, \rho^+(h^b(u))\}$.

UNION ($S(A), S(B)$): return $\{c_1, \dots, c_k\}$ where $c_i = \max\{a_i, b_i\}$.

SIZE ($S(A)$): return $\frac{\alpha_k \cdot k^2}{\sum_{j=1}^k 2^{-a_j}}$.

The constant α_k can be approximated to 0.72134 and is more precisely provided in Flajolet et al. (2007). The standard error is bounded by $1.04/\sqrt{k}$ using space $\log \log n + O(1)$ bits.

On Computing the Diameter

In this section we focus on the exact computation of the diameter in undirected graphs by using the i FUB algorithm, whose pseudocode is shown in Algorithm 2 (Crescenzi et al. 2013). First of all, recall that the *textbook* algorithm runs a BFS for any node, computing its eccentricity and finally returning the maximum eccentricity found. i FUB also computes the eccentricity of each node but processing the nodes in a specific order π . While doing this (i) it refines a lower bound lb (i.e., the maximum eccentricity found until that moment), and (ii) it refines an upper bound ub for the eccentricities of the remaining nodes. Finally, it stops when $ub \leq lb$ meaning that the remaining nodes cannot have eccentricity higher than our current lower bound.

Algorithm 2: *iFUB*

Input: A graph G , a node u , e.g., the node with highest degree.
Output: The diameter D

```

1  $i \leftarrow \text{ecc}(u)$ 
2  $lb \leftarrow \text{ecc}(u)$ 
3  $ub \leftarrow 2 \cdot \text{ecc}(u)$ 
4 while  $ub > lb$  do
5   let  $e$  be the maximum eccentricity of all nodes
      at distance  $i$  from  $u$ 
6    $lb \leftarrow \max\{lb, e\}$ 
7    $ub \leftarrow 2 \cdot (i - 1)$ 
8    $i \leftarrow i - 1$ 
9 return  $lb$ 
```

The order π is obtained sorting the nodes in decreasing order according to their distance from a given node, e.g., the highest degree node.

The time complexity of *iFUB* is in the worst case the same of the textbook algorithm, i.e., $O(nm)$, which is consistent with the time lower bound provided by Roditty and Williams (2013). This worst case is achieved when the nodes in the graph have close eccentricity or their BFSS' trees are isomorphic. However, the experiments in Crescenzi et al. (2013) have shown that in real-world graphs, this algorithm is almost always linear time, as the number of BFSS it performs seems to be constant. Moreover, an implementation of the algorithm in the WebGraph framework has computed the diameter of Facebook in 2011. At that time, Facebook network was composed by 721.1 millions of users and 68.7 billions of edges, and its diameter was computed by *iFUB* using 17 BFSs (Backstrom et al. 2012).

The above algorithm has been adapted to work for directed strongly connected graphs in Crescenzi et al. (2012). In Borassi et al. (2015) and Akiba et al. (2015), the algorithm has then been improved, in terms of practical performances, both for directed and undirected graphs, and it has been extended to directed weakly connected graphs, that is, directed graphs whose corresponding undirected graphs in which the direction of the edges is removed are connected.

Examples of Application**The six degrees of separation phenomenon.**

The *small-world experiment* was conducted by Stanley Milgram and other researchers in the 1960s in order to determine the average “social distance” between people in the United States. A package containing a letter, basic information about a target person, and an empty table on which the participants could write their name was sent to randomly selected individuals. Each participant was asked whether he/she personally knew the contact person. If so, the letter could be directly sent to that person. Otherwise the participant was asked to add his/her name on the table and to send the package to a person he/she knew personally who was more likely to know the target (who became a new participant of the experiment). In this way, for each package that reached the target person, by looking at the name table, it was possible to count the number of times it had been forwarded from person to person, that is, to estimate the social distance from the original participant to the target person. The observed average path length was less than or equal to six. For this reason, the researchers concluded that people in the United States were separated by about six “handshakes.” This hypothesis was so fascinating that a theater play was written in 1990, and later a film in 1993 popularized the play itself. The play is one of the main reasons why the small-world experiment has become popular as the *six degrees of separation* phenomenon. In the field of complex network analysis, this phenomenon has been tested on several different kinds of (online) social networks using the techniques presented in this entry. The HyperANF tool (Boldi et al. 2011) has been used to show that the degrees of separation in the Facebook network in 2011 (Backstrom et al. 2012) was four on the average instead of six.

Diameter computation. The diameter is a relevant measure (whose meaning depends on the semantics of the network itself), which

has been almost always considered while analyzing real-world networks such as biological, collaboration, communication, road, social, and web networks (Brandes 2005). Differently from the small-world experiment, in this case we are not interested on the average distance but on the worst-case distance. This information can be very useful in order to determine, for example, an upper bound on the number of rounds a specific protocol (like, for example, a flooding protocol) should run. Algorithm 1 is an example of such an application, as the diameter is the number of iterations needed to propagate neighborhood information inside the network and to stop the algorithm.

Software libraries. The *iFUB* algorithm and its improvements have been implemented in several graph algorithm libraries. These include Lasagne (<http://lasagne-unifi.sourceforge.net>), Sage (www.sagemath.org), NetworkKit (<http://networkkit.itい.Kit.edu>), and Webgraph (<http://webgraph.di.unimi.it>). The first library contains also the classical sampling algorithm for the distance distribution, while the last one includes the implementation of the HyperLogLog counters.

Future Directions for Research

The design of new sketching methods is currently an active area of research and, for instance, new estimators have been proposed in Cohen (2015). Moreover, future directions involve understanding why the *iFUB* works so well in the case of real-world networks despite its worst-case running time. Some steps in this direction have been done in Borassi et al. (2017).

Cross-References

► (Web/Social) Graph Compression

References

- Abboud A, Williams VV, Wang JR (2016) Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on discrete algorithms, SODA 2016, Arlington, 10–12 Jan, pp 377–391
- Akiba T, Iwata Y, Kawata Y (2015) An exact algorithm for diameters of large real directed graphs. In: Experimental algorithms – proceedings of 14th international symposium, SEA 2015, Paris, 29 June–1 July 2015, pp 56–67
- Backstrom L, Boldi P, Rosa M, Ugander J, Vigna S (2012) Four degrees of separation. In: Web science 2012, WebSci’12, Evanston, 22–24 June 2012, pp 33–42
- Boldi P, Vigna S (2003) The webgraph framework I: compression techniques. In: Proceedings of the 13th international world wide web conference. ACM, Manhattan, pp 595–601
- Boldi P, Rosa M, Vigna S (2011) Hyperanf: approximating the neighbourhood function of very large graphs on a budget. In: Proceedings of the 20th international conference on world wide web, WWW 2011, Hyderabad, 28 Mar–1 Apr 2011, pp 625–634
- Borassi M, Crescenzi P, Habib M, Kosters WA, Marino A, Takes FW (2015) Fast diameter and radius BFS-based computation in (weakly connected) real-world graphs: with an application to the six degrees of separation games. *Theor Comput Sci* 586:59–80
- Borassi M, Crescenzi P, Trevisan L (2017) An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. In: Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms, SODA 2017, Barcelona, Hotel Porta Fira, 16–19 Jan, pp 920–939
- Brandes U (2005) Network analysis: methodological foundations, vol 3418. Springer Science & Business Media, Berlin
- Brandes U, Pich C (2007) Centrality estimation in large networks. *Int J Bifurcation Chaos* 17(7):2303–2318
- Broder AZ (1997) On the resemblance and containment of documents. In: In compression and complexity of sequences (SEQUENCES’97). IEEE Computer Society, pp 21–29
- Cairo M, Grossi R, Rizzi R (2016) New bounds for approximating extremal distances in undirected graphs. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on discrete algorithms, SODA 2016, Arlington, 10–12 Jan 2016, pp 363–376
- Cohen E (1994) Estimating the size of the transitive closure in linear time. *Annu IEEE Symp Found Comput Sci* 35:190–200
- Cohen E (1997) Size-estimation framework with applications to transitive closure and reachability. *J Comput Syst Sci* 55(3):441–453

- Cohen E (2015) All-distances sketches, revisited: HIP estimators for massive graphs analysis. *IEEE Trans Knowl Data Eng* 27(9):2320–2334
- Cohen E, Kaplan H (2007a) Bottom-k sketches: better and more efficient estimation of aggregates. In: ACM SIGMETRICS performance evaluation review, vol 35. ACM, pp 353–354
- Cohen E, Kaplan H (2007b) Summarizing data using bottom-k sketches. In: PODC, pp 225–234
- Cohen E, Kaplan H (2008) Tighter estimation using bottom k sketches. *PVLDB* 1(1):213–224
- Crescenzi P, Grossi R, Imbrenda C, Lanzi L, Marino A (2010) Finding the diameter in real-world graphs: experimentally turning a lower bound into an upper bound. In: Proceeding ESA. LNCS, vol 6346, pp 302–313
- Crescenzi P, Grossi R, Lanzi L, Marino A (2011) A comparison of three algorithms for approximating the distance distribution in real-world graphs. In: Theory and practice of algorithms in (computer) systems – proceedings of first international ICST conference, TAPAS 2011, Rome, 18–20 Apr 2011, pp 92–103
- Crescenzi P, Grossi R, Lanzi L, Marino A (2012) On computing the diameter of real-world directed (weighted) graphs. In: Experimental algorithms – proceedings of 11th international symposium, SEA 2012, Bordeaux, 7–9 June 2012, pp 99–110
- Crescenzi P, Grossi R, Habib M, Lanzi L, Marino A (2013) On computing the diameter of real-world undirected graphs. *Theor Comput Sci* 514: 84–95
- Eppstein D, Wang J (2001) Fast approximation of centrality. In: Proceedings of the twelfth annual ACM-SIAM symposium on discrete algorithms, SODA'01. Society for Industrial and Applied Mathematics, Philadelphia, pp 228–229
- Flajolet P, Martin G (1985) Probabilistic counting algorithms for data base applications. *J Comput Syst Sci* 31(2):182–209
- Flajolet P, Éric Fusy, Gandouet O et al (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Proceedings of the 2007 international conference on analysis of algorithms, INAOFA'07
- Hoeffding W (1963) Probability inequalities for sums of bounded random variables. *J Am Stat Assoc* 58(301):13–30
- Impagliazzo R, Paturi R, Zane F (2001) Which problems have strongly exponential complexity? *J Comput Syst Sci* 63(4):512–530
- Magnien C, Latapy M, Habib M (2009) Fast computation of empirically tight bounds for the diameter of massive graphs. *J Exp Algorithmics* 13:1–10
- Palmer CR, Gibbons PB, Faloutsos C (2002) ANF: a fast and scalable tool for data mining in massive graphs. In: Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining, pp 81–90
- Roditty L, Williams VV (2013) Fast approximation algorithms for the diameter and radius of sparse graphs. In: Symposium on theory of computing conference, STOC'13, Palo Alto, 1–4 June 2013, pp 515–524

Delta Compression Techniques

Torsten Sue

Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Synonyms

Data differencing; Delta encoding; Differential compression

Definitions

Delta compression techniques encode a target file with respect to one or more reference files, such that a decoder who has access to the same reference files can recreate the target file from the compressed data. Delta compression is usually applied in cases where there is a high degree of redundancy between target and references files, leading to a much smaller compressed size than could be achieved by just compressing the target file by itself. Typical application scenarios include revision control systems and versioned file systems that store many versions of a file or software or content updates over networks where the recipient already has an older version of the data. Most work on delta compression techniques has focused on the case of textual and binary files, but the concept can also be applied to multimedia and structured data.

Delta compression should not be confused with Elias delta codes, a technique for encoding integer values, or with the idea of coding sorted sequences of integers by first taking the difference (or delta) between consecutive values. Also, delta compression requires the encoder to have complete knowledge of the reference files

and thus differs from more general techniques for redundancy elimination in networks and storage systems where the encoder has limited or even no knowledge of the reference files, though the boundaries with that line of work are not clearly defined.

Overview

Many applications of big data technologies involve very large data sets that need to be stored on disk or transmitted over networks. Consequently, data compression techniques are widely used to reduce data sizes. However, there are many scenarios where there are significant redundancies between different data files that cannot be exploited by compressing each file independently. For example, there may be many versions of a document, say a Wikipedia article, that differ only slightly from one to the next. Delta compression techniques attempt to exploit such redundancy between pairs or groups of files to achieve better compression.

We define delta compression for the most common case of a single reference file. Formally, we have two strings (files): $f_{tar} \in \Sigma^*$ (the target file) and $f_{ref} \in \Sigma^*$ (the reference file). Then the goal for an encoder E with access to both f_{tar} and f_{ref} is to construct a file $f_\delta \in \Sigma^*$ of minimum size, such that a decoder D can reconstruct f_{tar} from f_{ref} and f_δ . We also refer to f_δ as the *delta* of f_{tar} and f_{ref} .

Given this definition, research on delta compression techniques has focused on three challenges: (1) How to build delta compression tools that result in f_δ of small size while also achieving fast compression and decompression speeds and small memory footprint. (2) How to use delta compression techniques to compress collections of files, i.e., how to select suitable reference files given a target file, or how to select a sequence of delta compression steps between files to achieve good compression for a collection ideally while also allowing fast retrieval of individual files. (3) How to apply delta compression in various application scenarios.

Key Research Findings

String-to-String Correction and Differencing

Some early related work was done by Wagner and Fisher (1974), who studied the *string-to-string correction problem*. This is the problem of finding the best (shortest) sequence of insert, delete, and update operations that transform one string f_{ref} to another string f_{tar} . The solution in Wagner and Fisher (1974) is based on finding the longest common subsequence of the two strings using dynamic programming and adding all remaining characters to f_{tar} explicitly. However, the string-to-string correction problem does not capture the full generality of the delta compression problem. In particular, in the string-to-string correction problem, it is implicitly assumed that the data common to f_{tar} and f_{ref} appears in (roughly) the same order in the two files, and the approach cannot exploit the case of substrings in f_{ref} appearing repeatedly in f_{tar} .

This early work motivated Unix tools such as *diff* and *bdiff*, which create a *patch*, i.e., a set of edit commands, that can be used to update the reference file to the target file. Such tools are widely used to create patches for software updates, and one advantage is that these patches are easily readable by humans who need to understand the changes. However, the tools typically do not generate delta files of competitive size, due to (1) limitations in the type of edit operations that are supported, as discussed in the previous paragraph, and (2) the absence of native compression methods for reducing the size of the patch. The latter issue can be partially addressed by applying general purpose file compressors such as *gzip* or *bzip* to the generated patches, though this is still far from optimal.

The first problem was partially addressed by Tichy (1984), who defined the *string-to-string correction problem with block moves*. For a file f , let $f[i]$ denote the i th symbol of f , $0 \leq i < |f|$, and $f[i, j]$ denote the block of symbols from i until (and including) j . Then a *block move* is a triple (p, q, l) such that $f_{ref}[p, \dots, p + l - 1] = f_{tar}[q, \dots, q + l - 1]$, representing a nonempty common substring of f_{ref} and f_{tar} of length l .

The file f_δ can now be constructed as a minimal *covering set* of block moves, such that every $f_{tar}[i]$ that also appears in f_{ref} is included in exactly one block move.

Tichy (1984) also showed that a greedy algorithm results in a minimal cover set, leading to an f_δ that can be constructed in linear space and time using suffix trees. However, the multiplicative constant in the space complexity made the approach impractical. A more practical approach uses hash tables with linear space but quadratic time worst-case complexity (Tichy 1984). Subsequent work in Ajtai et al. (2002) showed how to further reduce the space used during compression while avoiding a significant increase in compressed size, while Agarwal et al. (2004) showed how to reduce size by picking better copies than a greedy approach. Finally, Xiao et al. (2005) showed bounds for delta compression under a very simple model of file generation where a file is edited according to a two-state Markov process that either keeps or replaces content.

Delta Compression Using Lempel-Ziv Coding

The block move-based approach proposed in Tichy (1984) leads to a basic shift in the development of delta compression algorithms, as it suggests thinking about delta compression as a sequence of copies from the reference file into the target file, as opposed to a sequence of edit operations on the reference file. This led to a set of new algorithms based on the Lempel-Ziv family of compression algorithms (Ziv and Lempel 1977, 1978), which naturally lend themselves to such a copy-based approach while also allowing for compression of the resulting patches.

In particular, the LZ77 algorithm can be viewed as a sequence of copy operations that replace a prefix of the string being encoded by a reference to an identical previously encoded substring. Thus, delta compression could be viewed as simply performing LZ77 compression with the file f_{ref} representing previously encoded text. In fact, we can also include the part of f_{tar} that has already been encoded in the search for a longest matching prefix. A few extra changes are needed to get a practical implementation of

LZ77-based delta compression. Implementations of this approach include *vdelta* (Hunt et al. 1998), several versions of *xdelta* (MacDonald 2000), *zdelta* (Trendafilov et al. 2002), the recent and very fast *ddelta* (Xia et al. 2014b) and *edelta* (Xia et al. 2015), and a number of implementations following the *vcdiff* differencing format.

A Sample Implementation

We now describe a possible implementation in more detail, using the example of *zdelta*. In fact, *zdelta* is based on a modification of the *zlib* library implementation of *gzip* by Gailly and Adler (Gailly 2017), with some additional ideas inspired by *vdelta* (Hunt et al. 1998). Note that *zlib* finds previous occurrences of substrings using a hash table. In *zdelta*, this is extended to several hash tables, one for each reference file and one for the already coded part of the target file. The table for f_{tar} is essentially handled as in *zlib*, where new entries are inserted as f_{tar} is traversed and encoded. The table for f_{ref} can be built beforehand by scanning f_{ref} , assuming f_{ref} is not too large. When looking for matches, all tables are searched to find the best match. Hashing of substrings is done based on the initial three characters, with chaining inside each hash bucket.

As observed in Hunt et al. (1998), in many cases, the location of the next match in f_{ref} is a short distance after the end of the previous match, especially when the files are very similar. Thus, the locations of matches in f_{ref} are usually best represented as offsets from the end of the previous match in f_{ref} . However, sometimes there are isolated matches in other parts of f_{ref} . This motivates *zdelta* to maintain two pointers into each reference file and to code locations by storing which pointer is used, the direction of the offset, and the offset itself. Pointers are initially set to the start of the file and afterward usually point to the ends of the previous two matches in the same reference file. If a match is far away from both pointers, a pointer is only moved to the end of it if it is the second such match in a row. Other more complex pointer movement policies might give additional moderate improvements.

To encode the generated offsets, pointers, directions, match lengths, and literals, *zdelta* uses

the Huffman coding facilities provided by *zlib*, while *vdelta* uses a byte-based encoding that is faster but less compact.

Window Management in LZ-Based Delta Compressors

The above description assumes that reference files are small enough to completely fit into the hash table. However, this is usually not realistic. By default, *zlib* only indexes a history of up to 32 KB of already processed data in the hash table, in two blocks of 16 KB each. Whenever the second block is filled, the first one is freed up, temporarily reducing indexed history to 16 KB. This is for efficiency reasons: Indexing more than 32 KB would result in larger hash tables that lead to increased L1 cache misses (as L1 cache sizes have not increased much over the last two decades) and, more importantly, result in very long hash chains for common three-character substrings that would be traversed when searching for matches.

In the target file, we could just index the last up to 32 KB in the hash table by their first 3 bytes, as done in *zlib*. But which parts of the reference files should we index? There are several possible choices, as follows:

- **Sliding Window Forward:** In *zdelta*, a 32 KB window is initially placed at the start of the reference file and then moved forward by 16 KB whenever the median of the last few copies comes within a certain distance from the end of the current window. This works well if the content is reasonably aligned between the reference and target case, which is the common case in many applications, but can perform very badly when large blocks of content occur in a very different order in the two files.
- **Block Fingerprints:** Another possible approach chooses the parts in the reference files that should be indexed based on their similarity to the content currently being encoded. This is done by computing fingerprint for certain substrings and then indexing parts of the reference file that share many fingerprints with the content we

are currently encoding. This approach was considered by Korn and Vo in the context of implementing *vcdiff* (Korn and Vo 2002).

- **Hashing Longer Substrings:** Another approach indexes the entire reference file and avoids long hash chains by hashing substrings much longer than three characters. This allows finding copies from anywhere in the reference and target files, as long as copies are longer than the hashed substrings. This approach was taken in earlier versions of *xdelta*, making it very robust against content reordering between the files, but compression may be slightly worse than other approaches on well-aligned files, since only large copies are supported. However, the approach can be combined with the others above, to allow both longer and shorter copies, by first using larger fixed-size chunks as proposed in Bentley and McIlroy (1999) and Tridgell (2000) or with content-defined chunking and then finding smaller copies more locally; see, e.g., *ddelta* (Xia et al. 2014b).

D

Compressing Collections of Files

Larger collections of files can be compressed by repeatedly applying delta compression to pairs or groups of files. This raises the question of what reference files to choose for which target files in order to achieve a cycle-free compression of the entire collection with minimum compressed size. In some scenarios, it is also crucial to be able to quickly decompress individual files without having to decompress too many other files.

In the case of revision control systems, older files are usually compressed using a newer version as a reference file, since newer versions are more frequently accessed. However, retrieving an old version could be very slow if it requires decompressing a long chain of more recent versions before reaching the one that is needed. For this reason, such systems may often create shortcuts, where versions are occasionally encoded using a much later version as reference, at the cost of a slight increase in size.

Delta compression can also be used to compress collections of non-versioned files that share

some degree of similarity. For the case of a single reference file, finding an optimal and cycle-free assignment of reference files to target files can be modeled as an *optimum branching* problem on a complete graph (Tate 1997; Ouyang et al. 2002), which can be solved in $|V|^2$ time on dense graphs where $|V|$ is the number of vertices. For the case of more than one reference file, the problem is known to be NP-complete (Adler and Mitzenmacher 2001). However, the cost of computing the edge weights of the input graph is prohibitive, motivating (Ouyang et al. 2002) to propose using text clustering to identify for each target file a small set of promising reference file candidates. Work in Bagchi et al. (2006) showed that an optimum branching on a reduced degree graph of highest weight edges in fact provides a provable approximation ratio to the solution on the full graph.

Work by Molfetas et al. (2014b,a) studied how to optimize and trade off access speed and compression ratio in delta-compressed file collections, e.g., by avoiding references to substrings in certain files when other copies could be used instead. It is shown that significant improvements are possible by judiciously picking copies during compression.

However, for bulk archival and retrieval of large collections of non-versioned web pages, work in Trendafilov et al. (2004), Chang et al. (2006), and Ferragina and Manzini (2010) suggests that better speed and compression can be achieved by applying optimized compression tools for archiving large data sets on top of a suitable linear ordering of the files, say one obtained via text clustering or URL ordering. The main reason is that such tools can identify repeated substrings over very long distances, while delta compressors are limited to one or a few reference files.

Examples of Applications

There are a number of scenarios where delta compression can be applied in order to reduce networking and storage cost. A few of them are now discussed briefly.

- **Software Revision Control Systems:** Delta compression techniques were originally proposed and developed in the context of systems used for maintaining the revision history of software projects and other documents (Berliner 1990; Rochkind 1975; Tichy 1985). In such systems, there are multiple, often almost identical, versions of each document, including different branches, that have to be stored, to enable users to retrieve and roll back to past versions. See Hunt et al. (1998) for more discussion on delta compression in the context of such systems.
- **Software Patch Distribution:** Delta compression techniques are used to generate software patches that can be efficiently transmitted over a network in order to update installed software packages. A good delta compressor can significantly reduce the cost of rolling out updates, say, for security-critical updates to popular software that need to be quickly disseminated. The case of updating automobile software over wireless links was recently discussed in Nakanishi et al. (2013), while Samteladze and Christensen (2012) addresses efficient app updates on mobile devices. Tools such as *bspatch* and *bsdiff* are especially optimized for the case of updating executables, and even better algorithms for this case are proposed in Percival (2006) and Motta et al. (2007).
- **Improving Data Downloading:** There is a long history of proposals to employ delta compression to improve web access, by exploiting the similarity between the current and an older cached version of a page or between different pages on the same site. A scheme called *optimistic delta* was proposed in Banga et al. (1997) and Mogul et al. (1997), where a caching proxy hides server latency by first returning a potentially outdated cached version, followed if needed by a small patch once the server replies. In another approach, a client with a cached old version sends a tag identifying the version as part of the HTTP request and then receives a patch (Housel and Lindquist 1996; Delco and Ionescu, xProxy: a transparent caching and delta transfer sys-

tem for web objects. Unpublished manuscript, 2000). A specialized tool called *jdelta* for XML data in push-based services is proposed in Wang et al. (2008)

It is well known that pages from the same web site are often very similar, mostly due to common layout and menu structure and that this could be exploited with delta compression. Work in Chan and Woo (1999) and Savant and Suel (2003) studies how to identify cached pages that make good reference files for delta compression, while Douglis et al. (1997) proposes a similar idea for shared dynamic pages, e.g., different stock quotes from a financial site.

While delta compression has the potential to significantly reduce latency and bandwidth usage, particularly over slower mobile links, such schemes have only seen limited adoption. Examples of widely used implementations are the *Shared Dictionary Compression over HTTP* mechanism implemented by Google in the Chrome browser and in their Brothli compressor (Alakuijala and Szabadka 2016) (which is based on the vcdiff differencing format and employs a mechanism similar to delta compression) and the use of delta compression for reducing network traffic in Dropbox (Drago et al. 2013).

- **Delta Compression in File and Storage Systems:** Delta compression is also useful for versioning file systems that keep old versions of files. The *Xdelta File System* (MacDonald 2000) aimed to provide efficient support for delta compression at the file system level using *xdelta*. Delta compression is also used for redundancy elimination in backup systems. In particular, Kulkarni et al. (2014), Shilane et al. (2012), and Xia et al. (2014b,a) showed that adding delta compression between similar files or chunks of data on top of duplicate elimination (removal of identical files or chunks) can give significant additional size reductions, though at some processing cost. Delta compression was implemented, e.g., in the EMC Data Domain line of products.
- **Efficient Storage of File Collections:** As discussed in the previous section, delta com-

pression has been considered as a way to improve compression of general collections of files that share some degree of similarity; see, e.g., Ouyang et al. (2002). However, subsequent work (Trendafilov et al. 2004; Chang et al. 2006; Ferragina and Manzini 2010) indicates that this approach is often outperformed by optimized compression tools for archiving large data sets, after suitable reordering of the files.

- **Exploring File Differences:** Delta compression tools can be used to visualize differences between documents. For example, the *diff* utility displays the differences between two files as a set of edit commands, while the *HtmlDiff* and *topblend* tools in Chen et al. (2000) visualize the difference between HTML documents. Here, the focus is less on compressed size and more on readability.

Future Directions for Research

Delta compression is a fairly mature technology, and future gains in compression may thus be modest for general scenarios. However, there are still a number of challenges that remain, including the following:

- **Speed Improvements:** Recent years have seen significant improvements in speed for many compression techniques, in some cases due to use of available data parallel (SIMD) instruction sets in modern processors. Examples of recent high-speed methods for delta compression are *ddelta* (Xia et al. 2014b) and *edelta* (Xia et al. 2015), and future work should further improve on these results.
- **Alternative Approaches:** Almost all current delta compressors use LZ-based approaches. While this is a natural approach, there might be alternative approaches (e.g., methods based on Burrows-Wheeler compression Burrows and Wheeler 1994) that could lead to improvements. On a more speculative level, researchers have recently started using recurrent neural networks for better compression, and such an approach might also lead to better delta compression.

- Formal Analysis of Methods:** Current delta compression algorithms are very heuristic in nature, and there is limited work on formally analyzing their performance or optimality with respect to information theoretic measures. An exception is the work in Xiao et al. (2005), which shows some bounds under a very simple model of document generation, but there is a need for more formal analysis.
- Integration in Storage Systems:** As discussed, delta compression techniques have been deployed inside storage systems in conjunction with other redundancy elimination techniques. In such systems, it can be challenging to identify suitable reference files for a target file that needs to be stored and to decide when it is worth to apply delta compression techniques, which require fetching the reference files for coding and decoding, in conjunction with other redundancy elimination techniques. Future research could look for new methods that use fingerprints and associated indexes to quickly find good reference files or explore better tradeoffs between compression and the amount of knowledge that is needed about the reference data (where delta compression is the case of full knowledge).

References

- Adler M, Mitzenmacher M (2001) Towards compressing web graphs. In: IEEE data compression conference
- Agarwal R, Amalapuram S, Jain S (2004) An approximation to the greedy algorithm for differential compression of very large files. In: IEEE data compression conference
- Ajtai M, Burns R, Fagin R, Long D, Stockmeyer L (2002) Compactly encoding unstructured inputs with differential compression. *J ACM* 49(3):318–367
- Alakuijala J, Szabadka Z (2016) Rfc7932: Brotli compressed data format. Available at <https://tools.ietf.org/html/rfc7932>
- Bagchi A, Bhargava A, Suel T (2006) Approximate maximum weighted branchings. *Inf Process Lett* 99(2): 54–58
- Banga G, Douglis F, Rabinovich M (1997) Optimistic deltas for WWW latency reduction. In: USENIX annual technical conference
- Bentley J, McIlroy D (1999) Data compression using long common strings. In: IEEE data compression conference
- Berliner B (1990) CVS II: Parallelizing software development. In: Winter 1990 USENIX conference
- Burrows M, Wheeler D (1994) A block-sorting lossless data compression algorithm. Technical report. 124, SRC. Digital Systems Research Center, Palo Alto
- Chan M, Woo T (1999) Cache-based compaction: a new technique for optimizing web transfer. In: INFOCOM conference
- Chang F, Dean J, Ghemawat S, Hsieh W, Wallach D, Burrows M, Chandra T, Fikes A, Gruber R (2006) Bigtable: a distributed storage system for structured data. In: Seventh symposium on operating system design and implementation
- Chen Y, Douglis F, Huang H, Vo K (2000) Topblend: an efficient implementation of HtmlDiff in Java. In: WebNet 2000 conference
- Douglis F, Haro A, Rabinovich M (1997) HPP: HTML macro-preprocessing to support dynamic document caching. In: USENIX symposium on internet technologies and systems
- Drago I, Bocchi E, Mellia M, Slatman H, Pras A (2013) Benchmarking personal cloud storage. In: Internet measurement conference
- Ferragina P, Manzini G (2010) On compressing the textual web. In: ACM international conference on web search and data mining
- Gailly J (2017) zlib compression library, version 1.2.11. Available at <https://zlib.net>
- Housel B, Lindquist D (1996) WebExpress: a system for optimizing web browsing in a wireless environment. In: ACM conference on mobile computing and networking, pp 108–116
- Hunt J, Vo KP, Tichy W (1998) Delta algorithms: an empirical analysis. *ACM Trans Softw Eng Methodol* 7:192–213
- Korn D, Vo KP (2002) Engineering a differencing and compression data format. In: USENIX annual technical conference, pp 219–228
- Kulkarni P, Douglis F, LaVoie J, Tracey JM (2014) Redundancy elimination within large collections of files. In: USENIX annual technical conference
- MacDonald J (2000) File system support for delta compression. MS thesis, University of California, Berkeley
- Mogul JC, Douglis F, Feldmann A, Krishnamurthy B (1997) Potential benefits of delta-encoding and data compression for HTTP. In: ACM SIGCOMM conference, pp 181–196
- Molfetas A, Wirth A, Zobel J (2014a) Scalability in recursively stored delta compressed collections of files. In: Second Australasian web conference
- Molfetas A, Wirth A, Zobel J (2014b) Using inter-file similarity to improve intra-file compression. In: IEEE international congress on big data
- Motta G, Gustafson J, Chen S (2007) Differential compression of executable code. In: IEEE data compression conference

- Nakanishi T, Shih H, Hisazumi K, Fukuda A (2013) A software update scheme by airwaves for automotive equipment. In: International conference on information, electronics, and vision
- Ouyang Z, Memon N, Suel T, Trendafilov D (2002) Cluster-based delta compression of a collection of files. In: Third international conference on web information systems engineering
- Percival C (2006) Matching with mismatches and assorted applications. PhD thesis, University of Oxford
- Rochkind M (1975) The source code control system. IEEE Trans Softw Eng 1:364–370
- Samteladze N, Christensen K (2012) Delta: delta encoding for less traffic for apps. In: IEEE conference on local computer networks
- Savant A, Suel T (2003) Server-friendly delta compression for efficient web access. In: 8th international workshop on web content caching and distribution
- Shilane P, Huang M, Wallace G, Hsu W (2012) WAN optimized replication of backup datasets using stream-informed delta compression. In: USENIX symposium on file and storage technologies
- Tate S (1997) Band ordering in lossless compression of multispectral images. IEEE Trans Comput 46(45): 211–320
- Tichy W (1984) The string-to-string correction problem with block moves. ACM Trans Comput Syst 2(4): 309–321
- Tichy W (1985) RCS: a system for version control. Softw Pract Exp 15:637–654
- Trendafilov D, Memon N, Suel T (2002) zdelta: a simple delta compression tool. Technical report. Polytechnic University, CIS Department
- Trendafilov D, NMemon, Suel T (2004) Compressing file collections with a TSP-based approach. Technical report TR-CIS-2004-02. Polytechnic University
- Tridgell A (2000) Efficient algorithms for sorting and synchronization. PhD thesis, Australian National University
- Wagner RA, Fisher MJ (1974) The string-to-string correction problem. J ACM 21(1):168–173
- Wang J, Guo Y, Huang B, Ma J, Mo Y (2008) Delta compression for information push services. In: International conference on advanced information networking and applications – workshops
- Xia W, Jiang H, Feng D, Tian L (2014a) Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In: IEEE data compression conference
- Xia W, Jiang H, Feng D, Tian L, Fu M, Zhou Y (2014b) Ddelta: a deduplication-inspired fast delta compression approach. Perform Eval 79:258–272
- Xia W, Li C, Jiang H, Feng D, Hua Y, Qin L, Zhang Y (2015) Edelta: a word-enlarging based fast delta compression approach. In: USENIX workshop on hot topics in storage and file systems
- Xiao C, Bing B, Chang GK (2005) Delta compression for fast wireless internet downloads. In: IEEE GlobeCom
- Ziv J, Lempel A (1977) A universal algorithm for data compression. IEEE Trans Inf Theory 23(3):337–343
- Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. IEEE Trans Inf Theory 24(5):530–536

D**Delta Encoding**► [Delta Compression Techniques](#)**Dictionary-Based Compression**► [Grammar-Based Compression](#)**Differential Compression**► [Delta Compression Techniques](#)**Dimension Reduction**

Kasper Green Larsen
Aarhus University, Aarhus, Denmark

Definitions

Dimensionality Reduction is a technique for taking a high-dimensional data set (data objects with many features/attributes) and replacing it with a much lower-dimensional data set while still preserving similarities between data objects. Dimensionality reduction is useful for reducing

the memory requirements for storing a data set, as well as for speeding up algorithms.

Overview

In modern algorithm design and data analysis, we often face very high-dimensional data. High-dimensional data comes in many forms. As examples, we can think of a 10-megapixel image as a point with ten million coordinates, one for each pixel. In this way, an image becomes a ten million-dimensional point. Another often encountered example arises when we wish to compare documents based on similarity. One way to do this is to simply count how many times each dictionary word occurs in a document and compare documents based on these counts. This yields a representation of a document as a point with one coordinate for each of the about 10^5 common English words. The distance between these high-dimensional points naturally captures the similarity of documents. The dimensionality grows even more dramatically when one compares documents based on so-called w -shingles (sets of w consecutive words in the text). With a typical choice of $w = 5$, the number of possible 5-shingles grows to $(10^5)^5 = 10^{25}$, and thus a document becomes an extremely high-dimensional point (albeit with few non-zero coordinates).

Dimensionality Reduction is an approach to “compressing” such high-dimensional points. Dimensionality reduction works by taking a high-dimensional point set and then mapping/embedding it to another point set in much lower-dimensional space. Said more formally, we take a set $X = \{p_1, \dots, p_n\}$ of d -dimensional points and map them to a set $f(X) = \{f(p_1), \dots, f(p_n)\}$ of m -dimensional points, hopefully with m much smaller than d . Since the points have much fewer coordinates after this *embedding*, they take up much less storage space. In addition to reducing the size of data sets, dimensionality reduction can also help speed up algorithms: The running time of many algorithms depends naturally on the dimensionality of the input data. If we reduce the

dimensionality, we also reduce the running time of such algorithms.

The difficulty in dimensionality reduction comes from the conflicting goal of also preserving the geometry of the point set. Preserving the geometry can have many interpretations, but roughly speaking, it means that we want close points to remain close and far points to remain far apart. This chapter surveys the most popular notion of dimensionality reduction, namely, dimensionality reduction under Euclidian distances. Here the goal is to preserve Euclidian distances between points while reducing the dimension as much as possible. For two points $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$ in d -dimensional space, their Euclidian distance is simply the generalization of distances in the plane to higher dimensions, that is, the distance from x to y is

$$\sqrt{\sum_{i=1}^d (y_i - x_i)^2}.$$

If we think of points as vectors, then the distance between vectors x and y is the so-called ℓ_2 -norm of the vector $y - x$. For a vector z , we use the notation $\|z\|_2 = \sqrt{\sum_{i=1}^d z_i^2}$ to denote the ℓ_2 -norm of z . Thus the distance from x to y equals $\|y - x\|_2$. When we say *dimensionality reduction* in this chapter, we thus think of the task of taking a set of d -dimensional points $X = \{p_1, \dots, p_n\}$ and mapping them to m -dimensional points $f(X) = \{f(p_1), \dots, f(p_n)\}$, such that the distance $\|f(p_i) - f(p_j)\|_2$ roughly equals the original distance $\|p_i - p_j\|_2$ for all pairs of points p_i, p_j in X .

Key Research Findings

The cornerstone result in dimensionality reduction is the famous Johnson-Lindenstrauss lemma due to Johnson and Lindenstrauss (1984). For any set $X = \{p_1, \dots, p_n\}$ of n points in d -dimensional space (denoted \mathbb{R}^d), and for any approximation factor $\varepsilon \in (0, 1/2)$,

the result says that there exists an embedding $f(X) = \{f(p_1), \dots, f(p_n)\}$, with each $f(p_i)$ in \mathbb{R}^m and $m = O(\varepsilon^{-2} \lg n)$, such that for all points $p_i, p_j \in X$, it holds that

$$\|f(p_j) - f(p_i)\|_2 \geq (1 - \varepsilon) \|p_j - p_i\|_2$$

and

$$\|f(p_j) - f(p_i)\|_2 \leq (1 + \varepsilon) \|p_j - p_i\|_2.$$

Let us take a minute to parse the result. When given a point set X in \mathbb{R}^d , it says that the “user” can specify an approximation factor $\varepsilon \in (0, 1/2)$. For that chosen approximation factor, it is possible to take the points X and embed them into only $m = O(\varepsilon^{-2} \lg n)$ dimensions while guaranteeing that the distance between any two points is only distorted by a multiplicative factor $(1 \pm \varepsilon)$. Thus if two points were close to each other before embedding, they will remain close afterward. Similarly, if two points were far apart before embedding, they will remain far apart. It is quite remarkable that the *target dimension* m doesn’t even depend on the original dimensionality of the point set. Furthermore, the dependency on the number of points in X is only logarithmic.

The original construction of Johnson and Lindenstrauss (1984) is not meant to be practical, and as you may have observed, the above only talks about the *existence* of an embedding $f(X)$. In practice, we naturally care about computing $f(X)$ efficiently. The following simple algorithm can be found in Dasgupta and Gupta (2003):

Embed(X):

- Let $m = \Theta(\varepsilon^{-2} \lg n)$. Form an $m \times d$ matrix A with each entry chosen independently and uniformly at random among $\{-1, +1\}$.
- Embed each point $x \in X$ to the point $f(x) = (Ax)/\sqrt{m}$ (compute Ax and divide all entries by \sqrt{m}).

Intuitively, the above algorithm chooses m random directions and projects the data onto those directions. The algorithm is randomized, and there is a small probability that it fails in the sense that some pair of points has their distance distorted by more than $(1 \pm \varepsilon)$. The failure probability is polynomially small in n when $m = C\varepsilon^{-2} \lg n$ for a big enough constant $C > 0$. If one cannot live with this small probability of failure, one can always run through all n^2 pairs of points and explicitly check if any pair have their distance distorted by too much. If that is the case, one can start over with a new random matrix A . Of course this is very time-consuming as we have to check n^2 pairs. Thus one would typically just hope for the best and not try to verify whether all distances were preserved. Finding an efficient deterministic construction remains an intriguing open problem (see Future Directions for Research).

Computing $f(x)$ for a data point $x \in X$ takes $O(md)$ time by a straightforward implementation of matrix-vector multiplication.

Remark 1 The paper by Dasgupta and Gupta (2003) actually has each entry of A a uniform random $\mathcal{N}(0, 1)$ distributed random variable. This makes their proof simpler but makes the construction less efficient when implemented in practice. For a correctness proof when using $-1/ + 1$ entries, see, e.g., Nelson (2015).

Remark 2 Much research has gone into determining whether the number of dimensions $m = O(\varepsilon^{-2} \lg n)$ is the best possible when given the desired approximation factor ε . This question was only answered very recently, where Larsen and Nelson (2017) showed that there exist point sets where one cannot embed into fewer than $\Omega(\varepsilon^{-2} \lg n)$ dimensions while preserving all pairwise distances to within $(1 \pm \varepsilon)$. Thus the Johnson-Lindenstrauss lemma is *optimal* when it comes to the target dimension m .

Speeding Up the Embedding

In many applications, the embedding time of $O(md)$ may not be sufficiently fast. For instance, if the original dimensionality is in the millions

and one embeds into a couple of thousand dimensions, then the matrix-vector multiplication takes in the order of billions instructions for each single data point. Several approaches have been suggested for speeding up the embedding time. We discuss the two main directions in the following:

Fast Johnson-Lindenstrauss Transforms. The first approach, due to Ailon and Chazelle (2009), replaces the matrix A from the previous algorithm with another carefully chosen matrix B , such that computing Bx can be done faster than computing Ax . To be precise, they actually replace Ax/\sqrt{m} by the product of three matrices PHD and x , i.e., $f(x) = PHDx$. The matrix D is a $d \times d$ diagonal matrix with each diagonal set independently and uniformly at random among $\{-1, +1\}$ (and all off-diagonals are 0). The matrix H is the $d \times d$ normalized Walsh-Hadamard matrix, where the entry $h_{i,j}$ is equal to $(-1)^{\langle i-1, j-1 \rangle}$ and $\langle i, j \rangle$ is the inner product modulo 2 of the vectors corresponding to the binary representations of i and j . Finally, P is an $m \times d$ matrix where the entries are chosen independently as follows: With probability $1-q$, set $p_{i,j}$ to 0 and otherwise let $p_{i,j}$ be $\mathcal{N}(0, 1/q)$ distributed. Here

$$q = \min \left\{ \Theta((\lg^2 n)/d), 1 \right\}.$$

While the embedding algorithm seems more complicated, observe first that computing Dx takes $O(d)$ time since D is a diagonal matrix. Applying H can be done in $O(d \lg d)$ time using the fast Fourier transform. Finally, P has an expected $qmd = O(m \lg^2 n) = O(\varepsilon^{-2} \lg^3 n)$ non-zeroes; thus applying P can be done in $O(m \lg^2 n)$ time, resulting in an embedding time of $O(d \lg d + m \lg^2 n)$. This may or may not be faster than the original embedding time of $O(md)$ depending on the relationship between m and d , but for most natural values of m and d , this would be preferred.

Sparse Matrices. The second approach replaces the matrix A with a much sparser matrix C . If

C has few non-zeroes, then computing Cx can again be done faster than computing Ax . Before we present the results, let us first make another comment on the time it takes to compute Ax in the first algorithm we presented. Here A was an $m \times d$ matrix with all entries either -1 or $+1$. If our input vector x has few non-zeroes, then computing Ax can also be done faster. To see this, note that Ax is simply $\sum_j x_j a_j$ where a_j is the j 'th column of A . All terms in this sum where $x_j = 0$ can be ignored. Thus if we let $\|x\|_0$ denote the number of non-zero coordinates of x , the embedding time actually improves to $O(m\|x\|_0)$. In many real-life applications (such as the example of documents and w -shingles earlier), the input vectors are very sparse, and thus exploiting the sparsity is crucial. Also observe that the fast Johnson-Lindenstrauss transform above (computing PHD) cannot exploit the sparsity of x as applying H using the fast Fourier transform takes time $O(d \lg d)$ regardless of the sparsity of Dx .

Kane and Nelson (2014) proposed the following way to choose the matrix C : Let $m = \Theta(\varepsilon^{-2} \lg n)$ and let $s = \Theta(\varepsilon^{-1} \lg n)$. Think of the m output dimensions as being partitioned into s chunks of m/s consecutive coordinates each. In each column of C , we choose one uniform random entry in each chunk and set it uniformly at random to either $-1/\sqrt{s}$ or $+1/\sqrt{s}$. All other entries are set to 0. They proved that letting $f(x) = Cx$ for each point $x \in X$ also preserves all pairwise distances to within $(1 \pm \varepsilon)$ with high probability. The embedding time improves to $O(s\|x\|_0)$ instead of $(m\|x\|_0)$, saving an ε^{-1} factor over the original approach.

Feature Hashing. Taking the sparse matrix approach to the extreme, Weinberger et al. (2009) proposed *feature hashing*. Their suggestion is to simply have one non-zero in each column of C , placed at a uniform random location and being uniform random among $\{-1, +1\}$. This of course gives the fastest possible embedding time of just $O(\|x\|_0)$. The downside is that the algorithm only works with high probability if the difference between any two input vectors is *spread out* on

many coordinates. More concretely, the current best analysis due to Dahlgaard et al. (2017) proves that feature hashing preserves all pairwise distances to within $(1 \pm \varepsilon)$ with high probability, as long as all pairwise differences $x - y$ satisfy that

$$\|x - y\|_\infty = O\left(\|x - y\|_2 \sqrt{\frac{\varepsilon \lg(1/\varepsilon)}{\lg n \lg(mn)}}\right).$$

Here $\|x - y\|_\infty$ is the largest absolute value of a coordinate in $x - y$. Thus feature hashing works as long as for any pairwise difference $x - y$, there is no single coordinate that is responsible for a large fraction of the distance between x and y . Feature hashing seems to work very well in practice and has been successfully applied in many applications.

Remarks In both feature hashing and the sparse matrix approach, we described the algorithms as first computing the matrix C and then computing Cx when embedding a point. In the introductory example with w -shingles, the input dimension d was 10^{25} , making it infeasible to store C explicitly. What one does instead is to use a *hash function* that allows you to compute the columns of C corresponding only to the non-zero coordinates of an input vector x . Thus C is never stored explicitly. We refer the reader to Kane and Nelson (2014) for how to implement a hash function for their sparse approach and to the paper by Dahlgaard et al. (2017) for suggestions and experiments on how to implement feature hashing in practice.

Examples of Application

Dimensionality reduction in the form presented in this chapter is of course only applicable in applications where Euclidian distance between data points is a natural similarity measure. For text documents, if we have a coordinate for each w -shingle (w consecutive words) counting how many times it occurs in the text, using Euclidian

distance as a similarity measure makes good sense.

Dimensionality reduction may also serve as a preprocessing step in nearest neighbor search applications. In (Euclidian) nearest neighbor search, we must store a set of high-dimensional points X , such that when given a query point q , we can efficiently find the point $x \in X$ that is closest (most similar) to q . An important property of all the above algorithms is that they are all based on sampling a random matrix *independently* of the data points. Thus if we are to store a set of high-dimensional points X for nearest neighbor search, what we can do instead is as follows: Sample a matrix A (using any of the distributions described earlier) and compute $f(x) = Ax$ for each $x \in X$. Store the embedded data set in a data structure for nearest neighbor search (on m -dimensional points instead of d -dimensional points). When we wish to answer a query q (find the point closest to q), we first compute $f(q) = Aq$ and then search for the nearest point $f(x)$ to $f(q)$. If this finds a point $f(x)$ of distance a from $f(q)$, we know that the original point x has distance at most $(1 + \varepsilon)a$ from q and thus is also very close to q . The crucial point here is that A was chosen independently of q (and X), and thus one can simply apply the standard analysis on the point set $X \cup \{q\}$ to conclude that the distance from q to any point in X is preserved to within $(1 \pm \varepsilon)$ with high probability. For more information on nearest neighbor search in high dimensions, see, e.g., the survey Andoni and Indyk (2008).

Another exciting application of dimensionality reduction is in k -means clustering. In k -means clustering, we are given a set of n points $X = \{p_1, \dots, p_n\}$ in d -dimensional space. The goal is to find k cluster centers $\mu_1, \dots, \mu_k \in \mathbb{R}^d$ such that

$$\sum_{p_i \in X} \min_j \|\mu_j - p_i\|_2^2$$

is minimized. That is, we have to find the set of k centers such that the sum of squared distances from each data point to its nearest center is minimized. Lloyd's algorithm and the k -means++ algorithm in Arthur and Vassilvitskii (2007) give

efficient heuristics for finding good clusterings (finding the optimal clustering is known to be NP-hard). These algorithms have a running time of $O(tknd)$ where t is the number of iterations of local improvements. Boutsidis et al. (2009) proposed that one first performs dimensionality reduction on the input data in order to speed up Lloyd’s algorithm and the k -means++ algorithm. More concretely, they proved that if the embedded point set preserves all distances to within $(1 \pm \varepsilon)$, the cost of all clusterings are also preserved to within $(1 \pm \varepsilon)$. Thus one can run clustering algorithms on $m = O(\varepsilon^{-2} \lg n)$ dimensional points and find almost as good clusterings as before. This reduces the running time from $O(tknd)$ to $O(tknm)$ plus the time for the embedding. The current state of the art in dimensionality reduction for k -means clustering can be found in Cohen et al. (2015) where bounds better than $m = O(\varepsilon^{-2} \lg n)$ can be obtained for small values of k .

Future Directions for Research

A number of important open problems remain. We will present a few in the following.

Best Sparsity Achievable. As we saw earlier, the current best approach using sparse matrices has $O(\varepsilon^{-1} \lg n)$ non-zeroes in each column of the embedding matrix C . Since the number of non-zeroes per column dictates the running time, finding even sparser (random) matrices that work for any point set is an important open problem. So far, we know that one must have at least $\Omega(\varepsilon^{-1} \lg n / \lg(1/\varepsilon))$ non-zeroes per column if one insists on embedding into the optimal number of dimensions $m = O(\varepsilon^{-2} \lg n)$, see Nelson and Nguyen (2013). Thus there remains a gap of $\lg(1/\varepsilon)$ between the upper and lower bounds.

Best Feature Hashing Guarantees. Feature hashing took the extreme approach and had only one non-zero entry per column of C . Since this is already more sparse than the above lower bound tells us is possible, this comes at the cost of only having any guarantees when no coordinates

are too large in any pairwise difference vector $x - y$. It remains an open problem whether the requirement

$$\|x - y\|_\infty = O\left(\|x - y\|_2 \sqrt{\frac{\varepsilon \lg(1/\varepsilon)}{\lg n \lg(mn)}}\right).$$

is necessary or if perhaps feature hashing works with even larger coordinates.

Fast Johnson-Lindenstrauss Transforms. We also studied the fast Johnson-Lindenstrauss transform and saw that it gave an embedding time of $O(d \lg d + m \lg^2 n)$. Can this be improved to something closer to $O(d + m)$? Or maybe $O((d + m) \lg d)$?

References

- Ailon N, Chazelle B (2009) The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM J Comput* 39(1):302–322
- Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun ACM* 51(1):117–122
- Arthur D, Vassilvitskii S (2007) k -means++: The advantages of careful seeding. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 1027–1035
- Boutsidis C, Mahoney MW, Drineas P (2009) Unsupervised feature selection for the k -means clustering problem. In: Proceedings of the 22nd international conference on neural information processing systems (NIPS), pp 153–161
- Cohen MB, Elder S, Musco C, Musco C, Persu M (2015) Dimensionality reduction for k -means clustering and low rank approximation. In: Proceedings of the 47th annual ACM symposium on theory of computing (STOC), pp 163–172
- Dahlgaard S, Knudsen MBT, Thorup M (2017, to appear) Practical hash functions for similarity estimation and dimensionality reduction. In: Proceedings of the 31st annual conference on neural information processing systems (NIPS)
- Dasgupta S, Gupta A (2003) An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct Algorithm* 22(1):60–65
- Johnson WB, Lindenstrauss J (1984) Extensions of Lipschitz mappings into a Hilbert space. *Contemp Math* 26:189–206
- Kane DM, Nelson J (2014) Sparser Johnson–Lindenstrauss transforms. *J ACM* 61(1):4:1–4:23

- Larsen KG, Nelson J (2017, to appear) Optimality of the Johnson-Lindenstrauss lemma. In: Proceedings of the 58th annual symposium on foundations of computer science (FOCS). See <http://arxiv.org/abs/1609.02094>
- Nelson J (2015) Johnson-lindenstrauss notes. <http://people.seas.edu/~minilek/madalgo2015/index.html>
- Nelson J, Nguyen HL (2013) Sparsity lower bounds for dimensionality reducing maps. In: Proceedings of the 45th annual ACM symposium on theory of computing (STOC), pp 101–110
- Weinberger K, Dasgupta A, Langford J, Smola A, Attenberg J (2009) Feature hashing for large scale multitask learning. In: Proceedings of the 26th annual international conference on machine learning (ICML), pp 1113–1120

Distance Distribution

- ▶ Degrees of Separation and Diameter in Large Graphs

Distance Estimation

- ▶ Similarity Sketching

Distant Supervision from Knowledge Graphs

Alisa Smirnova, Julien Audiffren, and Philippe Cudré-Mauroux
Exascale Infolab, University of Fribourg, Fribourg, Switzerland

Synonyms

- ▶ Self-supervised relation extraction

Overview

In this chapter, we discuss approaches leveraging distant supervision for relation extraction. We start by introducing the key ideas behind distant

supervision as well as their main shortcomings. We then discuss approaches that improve over the basic method, including approaches based on the at-least-one-principle along with their extensions for handling false negative labels, and approaches leveraging topic models. We also describe embeddings-based methods including methods leveraging convolutional neural networks. Finally, we discuss how to take advantage of auxiliary information to improve relation extraction.

D

Definitions

Definition 1 A *knowledge graph*, or knowledge base, is a semantic network defined as a set of triples (s, p, o) specifying that a node s (subject) is connected to another node o (object) by the property p . Sets of such triples form a directed graph, where nodes in the graph represent the subject and object in the triples and labeled edges represent the predicates connecting the subjects to the values. Generally, knowledge graphs can be seen as a collection of RDF triplets (see <https://www.w3.org/RDF/>).

Definition 2 *Distant supervision* is a technique to automatically annotate input data using information contained in the knowledge graph. Given a knowledge graph \mathcal{G} and a text corpus \mathcal{C} (i.e., a collection of texts), the key idea of distant supervision is to align \mathcal{G} to \mathcal{C} . More specifically, the idea is to first collect those sentences from the corpus \mathcal{C} that mention the entity pair (e_1, e_2) where both e_1 and e_2 exist in the knowledge graph \mathcal{G} . If a sentence mentions (e_1, e_2) and there exists one triple (e_1, r, e_2) in the knowledge graph, then the distant supervision approach *labels* this sentence as an instance (also called *mention*) of relation r . The task of extracting such triples from raw text is called *relation extraction*.

Basic Approach

The idea of using a knowledge graph as a source of labels for training data was first proposed

by Mintz et al. (2009) and relies on the following assumption:

Assumption 1 *If two entities participate in a relation, any sentence that contains those two entities might express that relation.*

For example, the following sentence:

South African entrepreneur **Elon Musk** is known for founding **Tesla Motors** and SpaceX.

mentions the tuple (*Elon Musk*, *Tesla Motors*). Assuming that the triple (*Elon Musk*, *created*, *Tesla Motors*) exists in the knowledge graph, the textual sentence is labeled with the relation *created* and can be used as training data for subsequent relation extractions. In this context, the set of sentences sharing the same entity pair is typically called a *bag of sentences*.

The relation extraction task can be considered as a classification problem, where the goal is to predict, for every entity pair, the relation it participates in from multiple classes. The classifier needs training data where every entity pair is represented as a *feature vector* (a binary representation of the input) and labeled with a corresponding relation. After learning on the training data, the testing step is performed, where the classifier predicts the relation labels for previously unseen entity pairs.

The transformation of the text corpus \mathcal{C} into a usable training set involves several steps illustrated in Fig. 1. The first step is a preprocessing step where classical natural language processing

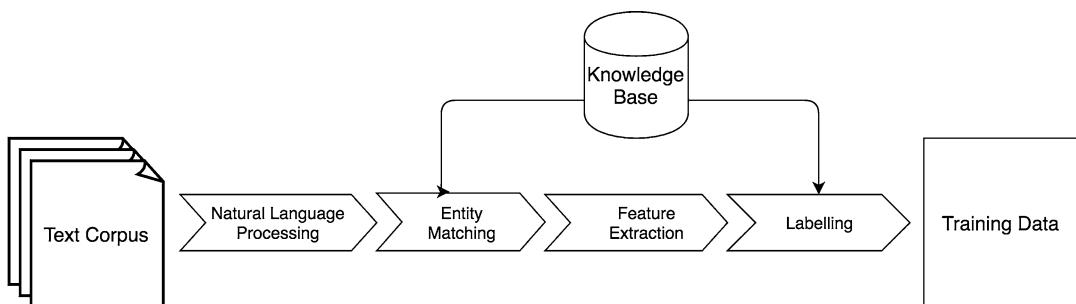
tools are used to identify potential entities from the text.

The second step, entity matching, takes as input the textual entities identified in the previous step and tries to match them to one of the instances in the knowledge graph (e.g., it tries to match “William Tell” as found in the text to its corresponding instance in Wikidata, e.g., instance number “Q30908” (see <https://www.wikidata.org/wiki/Q30908>)).

In the third step, sentences where two entities are correctly matched to the knowledge graph are processed to extract features. Two types of features, lexical and syntactic, were originally proposed by Mintz et al. (2009). Similar features were used in many subsequent approaches. Lexical features include the sequence of words between the two entities, flags indicating which entity name comes first in the sentence as well as the words immediately to the left of the first entity and to the right of the second entity. In addition to these features, syntactic features can be taken from a dependency parser – which extracts syntactic relations between words such as the dependency path between two entities.

Finally, the last step is called labeling, i.e., obtaining relation labels corresponding to the entity pair from the knowledge graph.

The test set is obtained as above when generating training data, except for the labeling part, i.e., the first three steps (natural language processing, entity matching, and feature extraction) remain the same. After the training and test sets are constructed, a standard classifier can be applied for relation extraction. In this context, Mintz



Distant Supervision from Knowledge Graphs, Fig. 1 The pipeline of preparing training data with distant supervision

et al. (2009) used a multi-class logistic classifier optimized using L-BFGS with Gaussian regularization.

Shortcomings of Distant Supervision

Automatically labeled training data are typically noisy, i.e., they contain false positives and false negatives, and this is also the case for distant supervision. On one hand, false negatives are mostly caused by the incompleteness of the knowledge graph. Two entities can be related in reality, but their relation might be missing in the knowledge graph; hence their mentions will be wrongly labeled as negative examples by distant supervision. On the other hand, two entities may appear in the same sentence because they are related to the same topic, but not necessarily because the sentence is expressing the relation. Such examples might yield false positives using distant supervision.

Distant Supervision Improvements

This section lists several extensions aimed at improving the basic distant supervision method.

At-Least-One Principle

Riedel et al. (2010) replace Assumption 1 with the following:

Assumption 2 *If two entities participate in a relation, at least one sentence that mentions those two entities will express that relation.*

This relaxed assumption, while more intuitive, comes at the cost of a more challenging classification problem. To tackle this problem, the authors propose an undirected graphical model that solves both the task of predicting a relation between two entities and the task of predicting which sentence expresses this relation. The model is developed on top of a *factor graph*, a popular probabilistic graphical network representation where an undirected bipartite graph connects relation *variables* with *factors* representing their joint probabilities.

The original model cannot capture the case when two entities participate in more than one relation. In our above example, the relation between *Elon Musk* and *Tesla Motors* is not only *co-founder_of* but also *CEO_of*. Hoffmann et al. (2011) and Surdeanu et al. (2012) propose undirected graphical models, called MultiR and MIML – RE, respectively, to perform *multi-instance multi-label* classification. Both models infer a relation expressed by a particular mention; thus the models are able to predict more than one relation between the two entities. The MultiR model is a conditional probability model that learns a joint distribution of both mention-level and bag-level assignments. The MIML – RE model contains two layers of classifiers. The first-level classifier assigns a label to a particular mention. The second level has k binary classifiers, where k is the number of known relation labels for the particular pair of entities. Each bag-level classifier y_i decides if the relation r_i holds for the given entity pair, using the mention-level classifications as input.

Negative Labels

In distant supervision, the knowledge graph only provides positive labels. Thus, negative training data is produced synthetically, e.g., by sampling the sentences containing two entities which are not related in the knowledge graph. The standard method to generate negative training data is to label mentions of unrelated entity pairs as negative samples. However, since the knowledge graph is incomplete, the absence of any relation label does not necessarily mean that the corresponding entities are not related. Hence, such a method potentially brings additional errors to the training data.

Several extensions of this model are proposed to overcome this problem. Min et al. (2013) propose an additional layer to MIML – RE in order to model the bag-level label noise. Ritter et al. (2013b) extend the MultiR model to handle missing data. Xu et al. (2013) explore ideas to enhance the knowledge graph, i.e., to infer entity pairs that are likely to participate in a relation even if they are unlabeled in the knowledge graph, and add them as positive samples.

Fan et al. (2014) propose an alternative approach to tackle the problems of both erroneous labeling and incompleteness of the knowledge graph. They formulate the relation extraction task as a *matrix completion* problem.

Topic Models

Another widely used approach to improve text analysis and text classification is topic models. In this context, topics represent clusters of terms (words or patterns) which often co-occur in the documents together. The goal of topic models is to assign a topic to every term.

Topic models can be applied to distant supervision by mapping a *document* to a sentence mentioning an entity pair and a *topic* to a relation. Words are represented by lexical and syntactic features, such as POS-tags or dependency paths between the entities. While the mention-level classifiers described above assign a relation to every sentence individually, the topic model classifiers are capable to capture more general dependencies between textual patterns and relations, which in practice can lead to improved performance.

Yao et al. (2011) propose a series of generative probabilistic models in that context. Though the presented models are designed for unsupervised, open relation extraction (i.e., the set of the target relations is not prespecified), the authors also show how the detected clusters can improve distantly supervised relation extraction. All their proposed models are based on *latent Dirichlet allocation* (LDA). The models differ in the set of features used and thus in their ability to cluster patterns. The advantage of generative topic models is that they are capable to “transfer” information from known patterns to unseen patterns, i.e., to associate a relation expressed by an already observed pattern to a new pattern.

Alfonseca et al. (2012) distinguish patterns that are expressing the relation from the ones that are not relation-specific and can be used across relations. For instance, the pattern “was born in” is relation-specific, while the pattern “lived in” can be used across different relations, i.e., *mayorOf* or *placeOfDeath*. Their proposed models are also based on LDA. Different submodels

have been suggested to capture three subsets of patterns:

- general patterns that appear for all relations;
- patterns that are specific for entity pairs and not generalizable across relations;
- patterns that are observed across most pairs with the same relation (i.e., relation-specific patterns).

Embedding-Based Methods

The methods discussed above are based on a large variety of lexical and syntactic features. We discuss below approaches that map the textual representation of relations and entity pairs onto a vector space. A mapping from discrete objects, such as words, to vectors of real number is called an embedding in this context. Embedding-based approaches for relation extraction do not require extensive feature engineering and natural language processing, but they still require NER tags to link entities in the knowledge graph with their textual mentions.

Riedel et al. (2013a) apply matrix factorization for the relation extraction task. The authors compute a low-rank factorization of the probability matrix M , where rows correspond to entity pairs and columns correspond to relations. The elements of the matrix correspond to the probability that a relation holds between two entities. This factorization provides an embedding of entity pairs and of relations, which is then used to predict the probability of new triplets using a logistic function and one of the following feature models:

1. The latent feature model (F) defines $\theta_{r,t}$ as a measure of compatibility between relation r and tuple t via the dot product of their embeddings a_r and v_t , respectively.
2. The neighborhood model (N) defines $\theta_{r,t}$ as a set of weights $w_{r,r'}$, corresponding to a directed association strength between relation r and r' , where both relations are observed for tuple t .

3. The entity model (E) also measures a compatibility between tuple t and relation r , but it provides latent feature vectors for every entity and every argument of relation r and thus can be used for n -ary relations. Entity types are implicitly embedded into the entity representation.
4. The combined model (NFE) defines $\theta_{r,t}$ as a sum of the parameters defined by the three above models.

Neural Networks

Another direction in text classification is applying convolutional neural networks (CNNs). CNNs are able to find specific n -grams in the text and classify the relations expressed in the sentences based on these n -grams.

Zeng et al. (2015) perform relation extraction via piecewise convolutional neural networks (PCNNs). The learning procedure of PCNNs is similar to standard CNNs. It consists of four parts: *vector representation, convolution, piecewise max pooling and softmax output*. In contrast to the embedding-based approaches above, the proposed model does not build word embeddings itself but uses pre-trained word vectors instead. Additionally, the model encodes a position of the word in the sentence with *position features* introduced by Zeng et al. (2014). Position features represent the relative distance between the current word in the sentence and both entities of interest e_1 and e_2 . The vector representation of a word is then the concatenation of word embedding and a position feature. In their work, the authors consider that the model predicts a relation for a bag (i.e., for an entity pair) if and only if a positive label is assigned to at least one entity mention.

Lin et al. (2016) extend the PCNN approach. The authors explore different methods to overcome the wrong labeling problem. More specifically, the model learns weights of the sentences in a bag in order to select sentences that are true relation mentions. Two ways of defining the weights are explored:

- Average, where every sentence has the same weight;

- Selective Attention, where sentence weight depends on the query-based function which scores how well the sentence expresses a given relation.

D Leveraging Auxiliary Information for Supervision

A number of distant supervision improvements use additional knowledge to enhance the model. Angeli et al. (2014) and Pershina et al. (2014) study the impact of extending training data with manually annotated data, which is generally speaking more reliable than the labels obtained from the knowledge graph. Pershina et al. (2014) propose to perform feature selection to generalize human labeled data into training guidelines and include them into the MIML – RE model. Angeli et al. (2014) propose to initialize the MIML – RE model with manually annotated data, since the original model is sensitive to initialization. As manual annotation is very expensive, carefully selecting which sentences to annotate is of utmost importance. The authors study several criteria to pick the most valuable sentences for manual annotation.

Another explored direction is improving entity identification, i.e., extract sentences mentioning entities not only by their canonical names but also by abbreviated mentions, acronyms, paraphrases, and even pronouns. Given the following sentences:

Elon Musk is trying to redefine transportation on earth and in space.
Through Tesla Motors – of which he is cofounder, CEO and Chairman – he is aiming to bring fully-electric vehicles to the mass market.

The second sentence contains a relation mention (Elon Musk, cofounderOf, Tesla Motors) that cannot be extracted without *co-reference resolution*, which refers to the task of clustering mentions of the same entity together, typically within a single sentence or document.

Augenstein et al. (2016) explore a variety of strategies for making entity identification tools more robust across domains. Moreover, they per-

form co-reference resolution for extracting relations across sentence boundaries. The approach is designed to perform relation extraction on very heterogeneous text corpuses, such as those extracted from the Web.

Koch et al. (2014) use entity types and co-reference resolution for a more accurate relation extraction. Two sets of entity types are available: coarse types (PERSON, LOCATION, ORGANIZATION, and MISC) come from the named entity recognizer (NER), while fine-grained types come from the knowledge graph. To bring type-awareness into the system, the authors build separate relation extractors on top of the MultiR model for each pair of coarse types, e.g., (PERSON, PERSON), (PERSON, LOCATION), and combine the extractions from the extractors of every type signature. Candidate mentions with incompatible entity types are then discarded, which improves precision at the cost of a slight drop in recall.

Chang et al. (2014) propose a tensor decomposition approach that also uses entity-type information and can be stacked with other models. In particular, it helps to overcome the fact that the matrix factorization model does not share information between rows, i.e., between entity pairs (including pairs containing the same entity).

Type-LDA is a topic model proposed by Yao et al. (2011) that considers fined-grained entity types based on latent Dirichlet allocation. In contrast to the previous approaches, it learns fine-grained entity types directly from the text.

Finally, a few relation extraction approaches integrate logical connections or constraints between the relations. As an example, the relation *capitalOf* between two entities directly implies another relation *cityOf*. Furthermore, many relation pairs are mutually exclusive, e.g., *spouseOf* and *parentOf*.

Rocktäschel et al. (2015) propose to inject logical formulae into the relations and entity pair embeddings. Their model is based on the matrix factorization approach from Riedel et al. (2013a). They explore two methods, namely, pre-factorization inference and joint optimization, and choose to focus on direct relation implication.

The approach proposed by Han and Sun (2016) is also based on the idea of using indirect supervision. The authors rely on a Markov logic network as a representation language and use:

- the consistency of relation labels, i.e., the interdependencies between relations that were mentioned previously (implications and mutual exclusion);
- the consistency between relation and arguments, i.e., entity-type information retrieved from the knowledge graph is used to filter inconsistent candidates;
- the consistency between neighboring instances: the authors define the similarity function between two relation candidates as the cosine similarity of their feature vectors.

Cross-References

- ▶ Deep Learning on Big Data
- ▶ Knowledge Graph Embeddings

References

- Alfonseca E, Filippova K, Delort JY, Garrido G (2012) Pattern learning for relation extraction with a hierarchical topic model. In: Proceedings of the 50th annual meeting of the association for computational linguistics: short papers-volume 2. Association for Computational Linguistics, pp 54–59
- Angeli G, Tibshirani J, Wu J, Manning CD (2014) Combining distant and partial supervision for relation extraction. In: EMNLP, pp 1556–1567
- Augenstein I, Maynard D, Ciravegna F (2016) Distantly supervised web relation extraction for knowledge base population. Semantic Web 7(4):335–349
- Chang KW, Yih SWt, Yang B, Meek C (2014) Typed tensor decomposition of knowledge bases for relation extraction. In: Proceedings of the 2014 conference on empirical methods in natural language processing, pp 1568–1579
- Fan M, Zhao D, Zhou Q, Liu Z, Zheng TF, Chang EY (2014) Distant supervision for relation extraction with matrix completion. In: ACL (1), Citeseer, pp 839–849
- Han X, Sun L (2016) Global distant supervision for relation extraction. In: Thirtieth AAAI conference on artificial intelligence
- Hoffmann R, Zhang C, Ling X, Zettlemoyer L, Weld DS (2011) Knowledge-based weak supervision for information extraction of overlapping relations. In:

- Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies-volume 1. Association for Computational Linguistics, pp 541–550
- Koch M, Gilmer J, Soderland S, Weld DS (2014) Type-aware distantly supervised relation extraction with linked arguments. In: Proceedings of EMNLP, Citeseer
- Lin Y, Shen S, Liu Z, Luan H, Sun M (2016) Neural relation extraction with selective attention over instances. In: ACL (1)
- Min B, Grishman R, Wan L, Wang C, Gondek D (2013) Distant supervision for relation extraction with an incomplete knowledge base. In: HLT-NAACL, pp 777–782
- Mintz M, Bills S, Snow R, Jurafsky D (2009) Distant supervision for relation extraction without labeled data. In: Proceedings of the joint conference of the 47th annual meeting of the ACL and the 4th international joint conference on natural language processing of the AFNLP: volume 2-volume 2. Association for Computational Linguistics, pp 1003–1011
- Pershina M, Min B, Xu W, Grishman R (2014) Infusion of labeled data into distant supervision for relation extraction. In: ACL (2), pp 732–738
- Riedel S, Yao L, McCallum A (2010) Modeling relations and their mentions without labeled text. Mach Learn Knowl Discovery Databases. Springer, pp 148–163
- Riedel S, Yao L, McCallum A, Marlin BM (2013a) Relation extraction with matrix factorization and universal schemas. Human language technologies: conference of the north american chapter of the association of computational linguistics, pp 74–84
- Ritter A, Zettlemoyer L, Etzioni O et al (2013b) Modeling missing data in distant supervision for information extraction. Trans Assoc Comput Linguist 1: 367–378
- Rocktäschel T, Singh S, Riedel S (2015) Injecting logical background knowledge into embeddings for relation extraction. In: HLT-NAACL, pp 1119–1129
- Surdeanu M, Tibshirani J, Nallapati R, Manning CD (2012) Multi-instance multi-label learning for relation extraction. In: Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning. Association for Computational Linguistics, pp 455–465
- Xu W, Hoffmann R, Zhao L, Grishman R (2013) Filling knowledge base gaps for distant supervision of relation extraction. In: ACL (2), pp 665–670
- Yao L, Haghghi A, Riedel S, McCallum A (2011) Structured relation discovery using generative models. In: Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics, pp 1456–1466
- Zeng D, Liu K, Lai S, Zhou G, Zhao J et al (2014) Relation classification via convolutional deep neural network. In: COLING, pp 2335–2344
- Zeng D, Liu K, Chen Y, Zhao J (2015) Distant supervision for relation extraction via piecewise convolutional neural networks. In: EMNLP, pp 1753–1762

Distributed Databases

► Robust Data Partitioning

D

Distributed Denial of Service (DDoS)

► Security and Privacy in Big Data Environment

Distributed File Systems

Yuchong Hu

School of Computer Science and Technology,
Huazhong University of Science and
Technology, Wuhan, China

Definitions

File system is a subsystem of an operating system which aims to organize, retrieve, and store data files. A distributed file system (DFS) is a file system with files shared on dispersed storage resources across a network. The DFS makes it convenient to share files among clients in a controlled and authorized manner, and the clients can benefit from DFS since they can locate all the file shares within a single server or domain name.

Overview

A wide variety of applications of big data analytics rely on distributed environments to analyze large amounts of data. As the amount of data increases, the need to provide reliable and efficient storage solutions has become one of the main concerns of big data infrastructure administrators. The traditional systems and methods of storage are suboptimal due to their price or performance restrictions, while DFS, as a paradigm shift in the storage arena, has been developed to

Distributed File Systems, Table 1 Types of transparencies

Transparency	Description
Access	Clients can access the files in a DFS in the same way as in a local file system
Location	Clients are unaware of the physical location of files
Concurrency	Clients have the same view of the state of the file system
Replication	Clients are unaware of replication of files across multiple servers
Migration	Clients are unaware of the movement of files across multiple servers

bring convenience to sharing files in both local area networks (LANs) and wide area networks (WANs).

The convenience is mainly brought by the technique of *transparency*. Transparency makes files accessed, stored, and managed on the local client machines, while the transparency process is actually held on the servers, such that the DFS implements its access control and storage management control to the client machines efficiently in a centralized way via the network file system, and one or more central servers store files that can be accessed by any client in the network. In other words, a DFS is a distributed implementation of the classical time-sharing model of a file system with multiple clients dispersed across the whole network of multiple and independent clients and makes it possible to restrict access to the file system depending on access lists or capabilities on both the servers and the clients.

The main goal of transparency in the DFS is to hide from the clients the fact that processes and resources are physically distributed across the network and provide a common view of a centralized file system. That is, DFS aims to be “invisible” to clients, who consider the DFS to be similar to a local file system. The different types of transparencies are listed in Table 1.

Key Points of Reliability

Reliability is of special significance in DFSes because the usage site of files can be different from

their storage site, so failure modes are substantially more complex in DFSes compared to local file systems. Replication and erasure coding are two typical techniques to achieve high reliability, and the following are key research findings based on the above two reliability schemes.

Replication Policy

DFSes use replication by making several copies of a data file on different servers. When a client requests the file, he transparently accesses one of the copies.

Replica placement plays a critical role in both performance and reliability of DFSes. Replicas are stored on different servers according to a placement scheme. Many DFSes (e.g., HDFS (Shvachko et al. 2010), GFS (Ghemawat et al. 2003)) use *random replication* as default, in which replicas are stored randomly on different nodes, on different racks, at different geographical locations, so that if a fault occurs anywhere in the system, data is still available. However, it cannot effectively separate the cluster into tiers while maintaining cluster durability and thus is quite susceptible to frequent data loss due to correlated failures. Facebook HDFS implementation (Borthakur et al. 2011) limits the placement of block replicas to smaller groups of nodes, thereby reducing the block loss probability with multiple node failures. Nonrandom replication schemes, like Copyset Replication (Cidon et al. 2013), improve Facebook scheme, by restricting the replicas to a minimal number of sets for a given scatter width. Scarlett (Ananthanarayanan et al. 2011) carefully stores replicas based on workload patterns to alleviate hotspots. Sinbad (Chowdhury et al. 2013) improves write performance by identifying the variance of link capacities in a DFS and avoiding storing replicas on nodes with congested links. Also, some studies address the efficient transitions between replication and erasure coding schemes (called *asynchronous encoding*) in DFSes. AutoRAID (Wilkes 1996) leverages access patterns to switch between replication for hot data and RAID-5 for cold data. DiskReduce (Fan et al. 2009) addresses the transition replication to erasure coding in

HDFS. EAR (Li et al. 2015) focuses on how replica placement affects the performance and reliability of asynchronous encoding.

In addition, replication disperses multiple copies of a file, and the changes have to be propagated to all the replicas in a consistent manner, especially in *cache consistency*, that is, how to update all copies of a data in cache when one of them is modified. The first method to ensure cache consistency is *write once read many* (WORM) (Quinlan 1991), in which cached files are in read-only mode (i.e., once a file is created, it cannot be modified) such that only the latest version of the data can be read. The second method is *transactional locking* (Dice et al. 2006), in which a read lock is obtained on the requested data file such that any other clients cannot write this file or a write lock in order is obtained to prevent any access on this file, allowing that each read reflects the latest write and each write is done in order. The third method is *leasing* (Gray and Cheriton 1989), in which the client is guaranteed that any other client cannot modify the data when the data file is requested during the lease. When the lease expires or the client releases its rights, the file is available again.

Erasure Coding

DFSes typically adopt replication as the fault tolerance scheme, while more and more studies propose and implement efficient erasure codes in DFSes for lower storage overhead. Zhang et al. (2010) apply erasure coding on the write path of HDFS to study the performance impact on various MapReduce (Dean and Ghemawat 2013) workloads. Silberstein et al. (2014) propose lazy recovery for erasure-coded storage to reduce repair bandwidth. Li et al. (2014) also propose degraded-first scheduling to improve scheduling of MapReduce jobs running on erasure-coded storage. HACFS (Xia et al. 2015), built on HDFS, dynamically switches encoded blocks between two erasure codes to balance storage overhead and repair performance. Repair pipelining (Li et al. 2017) speeds up the repair performance as a middleware system on HDFS. Many enterprises (e.g., Google (Ghemawat et al. 2003), Azure (Huang et al. 2012), and Facebook (Mu-

ralidhar et al. 2014)) have also deployed erasure coding in production DFSes to reduce storage overhead. Piggybacked-RS codes (Rashmi et al. 2013, 2014) embed parities of one stripe into that of the following stripe, thereby reducing repair bandwidth, which have also been evaluated in Facebook clusters. Facebook f4 (Muralidhar et al. 2014) protects failures at different levels including disks, nodes, and racks, by manipulating different redundancies of erasure coding.

Some studies propose new erasure code constructions and evaluate their applicability in DFSes. Local repairable codes (LRC) are a new family of erasure codes that reduce I/O during recovery by limiting the number of surviving nodes to be accessed. Azure's LRC (Huang et al. 2012) and Facebook's LRC (Sathiamoorthy et al. 2013) present and design a novel family of LRCs that are efficiently repairable based on a trade-off between locality and minimum distance.

Regenerating codes (Dimakis et al. 2007) are a state-of-the-art family of erasure codes that achieve the optimal trade-off between storage redundancy and repair traffic in DFSes. Constructions of regenerating codes have been proposed and implemented in DFSes, such as FMSR codes (Hu et al. 2012, 2017a), PM-RBT codes (Rashmi et al. 2015), and Butterfly codes (Pamies-Juarez et al. 2016). CORE (Li et al. 2015) extends regenerating codes to support the optimal recovery of multi-node failures atop HDFS. DoubleR (Hu et al. 2017b) proposes and implements new regenerating code constructions for hierarchical data centers atop HDFS.

Examples of Application

DFSes are the foundation of storage mechanisms of big data applications. Google GFS (Ghemawat et al. 2003) is an expandable DFS to support large-scale and data-intensive applications, using commodity servers to provide data reliability and high-performance services. In addition, there are other solutions to achieve different demands for storage of big data. For example, HDFS (Shvachko et al. 2010) is a derivative of open-source codes of GFS. Microsoft uses Cosmos

(Chaiken et al. 2008) to support its search and advertisement business. Facebook uses Haystack (Beaver et al. 2010) to store a large number of photos.

DFSes have been relatively mature after years of development, and it is hard to make a complete study given the number of existing DFSes. This section chooses to study two popular DFSes, HDFS (Shvachko et al. 2010) and Ceph (Weil et al. 2006a), from four aspects: architecture, client access, replication, and fault detection.

HDFS

Architecture: An HDFS cluster consists of a single *namenode*, which is the master node that manages the file system namespace and regulates the access to files by clients. The data stream is divided into blocks distributed across *datanodes*, which manage the storage attached to the nodes that they run on. HDFS provides a secondary namenode as a persistent copy of the namenode to handle namenode failures by restoring the namespace from the secondary namenode. The namenode executes different file system operations like opening, closing, and renaming files and directories. The datanodes serve clients' read or write requests and also perform block creation, deletion, and replication from the namenode.

Client Access: HDFS uses a code library that allows clients to read, write, and delete file and create and delete directories. Clients use a file's path in the namespace to access it, contact the namenode to get or put files' blocks, and do the transfer by connecting directly with the datanodes.

Replication Policy: HDFS divides data into blocks which are replicated across datanodes according to a placement policy, by which each datanode holds at most one copy of a block and each rack holds at most two copies of the block. The namenode verifies that each block follows the above policy: (1) when a block is over-replicated, the namenode removes a replica on the datanode that has the least amount of available space and tries not to reduce the number

of racks; (2) when a block is under-replicated, the namenode creates a new replica and places it in a priority queue which is verified periodically; and (3) when all replicas of a block are on the same rack, the namenode will add a new replica on a different rack to trigger the over-replicated policy.

Fault Management: Each datanode compares the software version and the namespace ID with those of the namenode. If they don't match, the datanode shuts down to preserve the integrity of the system. Datanodes do a registration with the namenode whenever they restart with a different IP address or port. After this registration and every hour, datanodes tell a current view of the location of each block to the namenode. Every 3 s, datanodes send heartbeats to the namenode, and the namenode considers datanodes as failed if it does not receive any heartbeats during 10 min.

Ceph

Architecture: Ceph uses metadata servers (MDS) to provide a dynamic distributed metadata management by performing queries of metadata and managing the namespace. Ceph stores data and metadata in object storage devices (OSDs) which mainly perform I/O operations. Ceph stripes each data into blocks which are assigned to objects, which are identified by the same inode number plus an object number and placed on storage devices using a function called CRUSH (Weil et al. 2006b).

Client Access: Ceph uses MDSs to get the inode number and the file size for client requests. Then, the client can calculate how many objects comprise the file and contacts the OSDs to get the data via CRUSH.

Replication Policy: Ceph stores data as objects which are put on different placement group (PG) and then stored on OSDs, which are chosen by the CRUSH. Ceph has three synchronous replication strategies, primary-copy, chain, and splay replication: (1) primary-copy replication updates all replicas in parallel; (2) chain replication updates

replicas in series; and (3) splay replication combines the parallel updates of primary-copy replication with the serial updates of chain replication.

Fault Management: Each OSD periodically exchange heartbeats to detect failures. Ceph has monitors that implement management of the cluster map. Each OSD can send a failure report to any monitor. When OSDs fail, monitors detect and maintain a valid cluster map.

Future Directions for Research

Although DFSes (e.g., HDFS) have become a mainstay in big data analytics file system platform, there are still improvements to be made. First, many DFSes (e.g., GFS and HDFS) originally designed for large files cannot meet the storage requirements of lots of small files, which will lead to the production of a large number of metadata, affect the metadata server management and restorability, and decline the entire system performance. Second, many DFSes have multiple copies (six copies in many cases) due to the need for data locality in maintaining performance, which makes it harder for already “big” data to store. Third, many DFSes often offer very limited SQL support and lack basic SQL functions which are actually useful for big data analytics. Finally, some DFSes for big data storage organize thousands or even hundreds of thousands of servers, which incurs substantial energy consumption, and thus the deployment of DFS should consider its energy efficiency.

Cross-References

► [Hadoop](#)

References

Ananthanarayanan G, Agarwal S, Kandula S, Greenberg A, Stoica I, Harlan D, Harris E (2011) Scarlett: coping with skewed content popularity in MapReduce clusters. In: European conference on computer systems, pro-

- ceedings of the sixth European conference on computer systems (EUROSYS 2011), Alzburg, pp 287–300
- Beaver D, Kumar S, Li HC, Sobel J, Vajgel P (2010) Finding a needle in haystack: Facebook’s photo storage. In: Usenix conference on operating systems design and implementation, pp 47–60
- Borthakur D, Gray J, Sarma JS, Muthukaruppan K, Spiegelberg N, Kuang H, Ranganathan K, Molkov D, Menon A, Rash S (2011) Apache Hadoop goes realtime at Facebook. In: ACM SIGMOD international conference on management of data (SIGMOD 2011), Athens, pp 1071–1080
- Chaiken R, Jenkins B, Ramsey B, Shakib D, Weaver S, Zhou J (2008) Scope: easy and efficient parallel processing of massive data sets. Proc VLDB Endow 1(2):1265–1276
- Chowdhury M, Kandula S, Stoica I (2013) Leveraging endpoint flexibility in data-intensive clusters. ACM SIGCOMM Comput Commun Rev 43(4):231–242
- Cidon A, Rumble SM, Stutsman R, Katti S, Ousterhout J, Rosenblum M (2013) Copysets: reducing the frequency of data loss in cloud storage. In: Usenix conference on technical conference, pp 37–48
- Dean J, Ghemawat S (2013) MapReduce: simplified data processing on large clusters. In: Proceedings of operating systems design and implementation (OSDI) 51(1):107–113
- Dice D, Shalev O, Shavit N (2006) Transactional locking II. In: International conference on distributed computing, pp 194–208
- Dimakis AG, Godfrey PB, Wainwright MJ, Ramchandran K (2007) Network coding for distributed storage systems. In: IEEE international conference on computer communications (INFOCOM 2007). IEEE, pp 2000–2008
- Fan B, Tantisiriroj W, Xiao L, Gibson G (2009) Diskreduce: raid for data-intensive scalable computing. In: The workshop on Petascale data storage, pp 6–10
- Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. ACM SIGOPS Oper Syst Rev 37(5):29–43
- Gray C, Cheriton D (1989) Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. ACM SIGOPS Oper Syst Rev 23(23):202–210
- Hu Y, Chen HCH, Lee PPC, Tang Y (2012) Nccloud: applying network coding for the storage repair in a cloud-of-clouds. In: Usenix conference on file and storage technologies, pp 12–19
- Hu Y, Lee PPC, Shum KW, Zhou P (2017a) Proxy-assisted regenerating codes with uncoded repair for distributed storage systems. IEEE Trans Inf Theory PP(99):1–17
- Hu Y, Li X, Zhang M, Lee PPC, Zhang X, Zhou P, Feng D (2017b) Optimal repair layering for erasure-coded data centers: from theory to practice. IEEE Trans Storage PP(99):1–26
- Huang C, Simitci H, Xu Y, Ogus A, Calder B, Gopalan P, Li J, Yekhanin S (2012) Erasure coding in windows azure storage. In: Usenix conference on technical conference, pp 2–13
- Li R, Lee PPC, Hu Y (2014) Degraded-first scheduling for MapReduce in erasure-coded storage clusters. In:

- IEEE/IFIP international conference on dependable systems and networks, pp 419–430
- Li R, Hu Y, Lee PPC (2015) Enabling efficient and reliable transition from replication to erasure coding for clustered file systems. *IEEE Trans Parallel Distrib Syst* PP(99):1–14
- Li R, Li X, Lee PPC, Huang Q (2017) Repair pipelining for erasure-coded storage. In: Usenix technical conference
- Muralidhar S, Lloyd W, Roy S, Hill C, Lin E, Liu W, Pan S, Shankar S, Sivakumar V, Tang L (2014) f4: Facebook’s warm blob storage system. In: Usenix conference on operating systems design and implementation, pp 383–398
- Pamies-Juarez L, Blagojevic F, Mateescu R, Gyuot C, Gad EE, Bandic Z (2016) Opening the chrysalis: on the real repair performance of MSR codes. In: International cultural heritage informatics meeting, pp 93–106
- Quinlan S (1991) A cached worm file system, vol 21. Wiley Online Library
- Rashmi KV, Shah NB, Gu D, Kuang H, Borthakur D, Ramchandran K (2013) A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster. *Usenix Hotstorage*
- Rashmi KV, Borthakur D, Borthakur D, Borthakur D, Borthakur D, Ramchandran K (2014) A “Hitchhiker’s” guide to fast and efficient data reconstruction in erasure-coded data centers. In: ACM conference on SIGCOMM, pp 331–342
- Rashmi KV, Nakkiran P, Wang J, Shah NB, Ramchandran K (2015) Having your cake and eating it too: jointly optimal erasure codes for i/o, storage and network-bandwidth. In: Usenix conference on file and storage technologies, pp 81–94
- Sathiamoorthy M, Asteris M, Papailiopoulos D, Dimakis AG, Vadali R, Chen S, Borthakur D (2013) Xoring elephants: novel erasure codes for big data. *Proc VLDB Endow* 6(5):325–336
- Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: MASS storage systems and technologies, pp 1–10
- Silberstein M, Ganesh L, Wang Y, Alvisi L, Dahlin M (2014) Lazy means smart: reducing repair bandwidth costs in erasure-coded distributed storage, pp 1–7
- Weil SA, Brandt SA, Miller EL, Long DDE, Maltzahn C (2006a) Ceph: a scalable, high-performance distributed file system. In: Symposium on operating systems design and implementation, pp 307–320
- Weil SA, Brandt SA, Miller EL, Maltzahn C (2006b) Crush: controlled, scalable, decentralized placement of replicated data. In: SC 2006 conference, proceedings of the ACM/IEEE, pp 31–31
- Wilkes J (1996) The HP autoraid hierarchical storage system. *ACM Trans Comput Syst* 14(1):108–136
- Xia M, Saxena M, Blaum M, Pease DA (2015) A tale of two erasure codes in HDFS. In: Usenix conference on file and storage technologies, pp 213–226
- Zhang Z, Deshpande A, Ma X, Thereska E, Narayanan D (2010) Does erasure coding have a role to play in my data center? Microsoft research MSR-TR-2010
-
- ## Distributed Incremental Query Computation
- ▶ [Distributed Incremental View Maintenance](#)
-
- ## Distributed Incremental View Maintenance
- Milos Nikolic
University of Oxford, Oxford, UK
- ### Synonyms
- [Distributed Incremental Query Computation](#); [Distributed View Update](#)
- ### Definitions
- A database view is a virtual table defined by a query over the given data sources. Database systems recompute the contents of a view upon each reference to its name. A materialized view precomputes and stores the result of its definition query. Incremental view maintenance keeps the contents of materialized views up-to-date for changes in the source relations. Distributed incremental view maintenance is the procedure of updating materialized views when the data is distributed at multiple sites in a computer network.
- ### Overview
- Modern applications require real-time analytics over changing datasets. In a growing number of domains – Internet of Things (IoT), clickstream

analysis, algorithmic trading, network monitoring, and fraud detection to name a few – applications monitor streaming data to promptly detect certain patterns, anomalies, or future trends. Such applications implement their logic using queries, or *views*, that continuously provide up-to-date results as the underlying data changes.

Providing fresh analytical results by recomputing views from scratch on every change is almost always inefficient. Since most datasets evolve through small changes, storing (materializing) previously computed results and combining them with incoming changes can yield computationally cheaper methods for updating query results. Maintaining the contents of a materialized view up to date for changes in its base relations is known as *incremental view maintenance* (IVM). Materialized views and incremental view maintenance are critical for speeding up the execution of frequently asked queries in database systems (e.g., Oracle, SQL Server, and DB2 all support incremental view maintenance) and achieving high-throughput, low-latency processing in stream management systems (Murray et al. 2013; Chandramouli et al. 2014; Zaharia et al. 2013; Nikolic et al. 2016). Chirkova and Yang (2012) survey the topic of materialized views.

Emerging data-intensive applications also demand scalable systems for querying and managing large datasets. In applications with large stateful working sets, like real-time data warehouses, distributed execution aims to speed up query evaluation and accommodate growing memory requirements. Scalable behavior is also essential when processing many concurrent activities like in IoT applications, which typically run the same query logic over a large collection of signals from sensor devices.

This chapter studies the topic of incremental view maintenance in distributed environments. It starts with an overview of incremental computation in local settings and then presents a technique for transforming local maintenance programs into data-parallel processing tasks. This technique applies a set of optimizations to speed up the maintenance task and minimize network overheads during distributed execution.

Incremental View Maintenance

Materialized views require maintenance to keep their contents up to date with changes in base tables. Refreshing materialized views using recomputation is expensive for frequent, small-sized updates. In such cases, applying only incremental changes, or *deltas*, to materialized views is often more efficient than recomputing the views from scratch.

Consider a materialized view represented as a pair $(Q, M(\mathbf{D}))$, where Q is the view definition query expressed in a declarative language such as SQL and $M(\mathbf{D})$ is the materialized contents of Q on a given database \mathbf{D} . When the database changes from \mathbf{D} to $(\mathbf{D} + \Delta\mathbf{D})$, where $\Delta\mathbf{D}$ can contain inserts, deletes, and updates to base tables, incremental view maintenance evaluates a delta query ΔQ to refresh $M(\mathbf{D})$.

$$M(\mathbf{D} + \Delta\mathbf{D}) = M(\mathbf{D}) + \Delta Q(\mathbf{D}, \Delta\mathbf{D})$$

Classical incremental view maintenance computes ΔQ on every input change $\Delta\mathbf{D}$. The delta ΔQ involves typically smaller delta tables instead of large base tables, thus computing ΔQ and updating $M(\mathbf{D})$ becomes cheaper than reevaluating Q from scratch. Incremental view maintenance derives one delta query for each referenced base table. The derivation process relies on a set of change propagation rules defined for each operator of the view definition language. Each derived delta query takes its role in the associated view maintenance trigger.

Example 1 Consider query Q that counts the tuples in the natural join of $R(A, B)$, $S(A, C, D)$, and $T(A, C, E)$.

```
Q = SELECT SUM(1)
    FROM R NATURAL JOIN S
        NATURAL JOIN T
```

For succinctness, the query is written as $Q = \text{Sum}_{[]} (R \bowtie S \bowtie T)$, where $[]$ denotes no group by variables. Let M_Q be the materialized view storing the result of Q . Incremental view maintenance keeps the contents of M_Q fresh for changes in the base relations. For instance, the

maintenance trigger handling a set of insertions ΔR into R is (the schemas are omitted for clarity):

```
ON UPDATE R BY ΔR:
M_Q += Sum[ ](ΔR ⋈ S ⋈ T)
R += ΔR
```

This trigger first increments the count by computing the delta query for updates to R and then updates the contents of R . In practice, ΔR is often much smaller than R , making incremental computation cheaper (faster) than reevaluation.

□

Data Model. Storing materialized views as generalized multiset relations (Koch 2010; Green et al. 2007) greatly simplifies the view maintenance task. Under this data model, each relation is treated as a map (dictionary) containing a finite number of unique tuples (keys) mapped to a non-zero payload (value). Such relations can uniformly represent insertions and deletions as tuples mapping to positive and, respectively, negative payloads (multiplicities). This unified representation admits simpler, more efficient view maintenance where insertions and deletions are commutative (i.e., applicable to the database in any order) and follow the same set of delta rules, which contrasts with the view maintenance under classical set and bag (multiset) semantics (Qian and Wiederhold 1991; Griffin and Libkin 1995).

This unified model also allows using tuple payloads for storing (potentially non-integer) aggregate values (e.g., SUM, AVG). Traditional SQL represents these aggregates as separate columns in the query result, while this model keeps them in the payload to facilitate incremental processing – updating aggregate values means changing only tuple payloads rather than deleting and inserting tuples from the result.

Query Language. The query language for generalized multiset relations consists of union $+$, natural join \bowtie , selection σ_θ , and grouping sum aggregation $\text{Sum}_{A;f}$. These operations naturally generalize those on multiset relations: for instance, $Q_1 \bowtie Q_2$ matches tuples of Q_1 with tuples of Q_2 on their common columns, multiplying their payloads (multiplicities); the SQL equivalent of $\text{Sum}_{A;f} R$ is `select A, SUM(f)`

from R group by A , except that the aggregate value is stored inside the payload of the group by tuples; for clarity, $\text{Sum}_A R$ is used when $f = 1$. This fragment of the language suffices to express flat queries with sum aggregates; Koch et al. (2014) further extend this language to support queries with nested aggregates.

Delta Queries. Delta queries capture changes in query results for updates to base relations. For any query expression Q expressed as an operator tree, its delta query ΔQ is constructed by applying delta derivation rules to the tree in a bottom-up fashion. If Q represents a relation R , then $\Delta Q = \Delta R$ if there are updates to R and $\Delta Q = 0$ otherwise. The delta rules for the other operators are:

$$\begin{aligned}\Delta(Q_1 + Q_2) &:= \Delta Q_1 + \Delta Q_2 \\ \Delta(Q_1 \bowtie Q_2) &:= (\Delta Q_1 \bowtie Q_2) + (Q_1 \bowtie \Delta Q_2) \\ &\quad + (\Delta Q_1 \bowtie \Delta Q_2) \\ \Delta(\sigma_\theta Q) &:= \sigma_\theta(\Delta Q) \\ \Delta(\text{Sum}_{A;f} Q) &:= \text{Sum}_{A;f}(\Delta Q)\end{aligned}$$

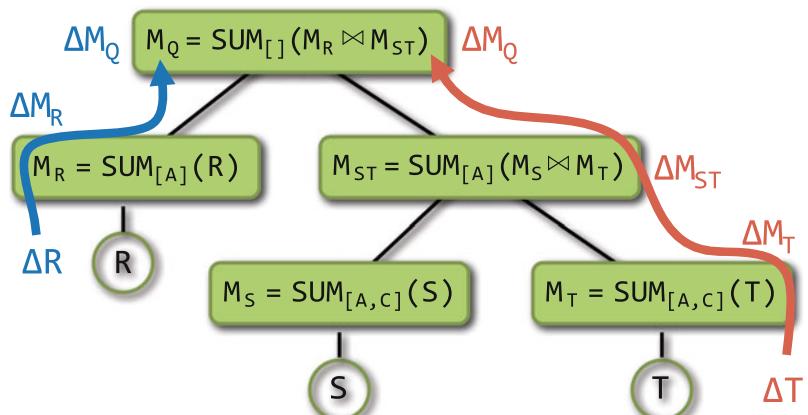
Previous work (Koch 2010; Koch et al. 2014) studies these rules in detail.

Higher-Order IVM. Incremental view maintenance is often cheaper than naïve reevaluation but is not free. Computing deltas can be expensive, like in Example 1, where ΔQ is a nontrivial join of one (small) input update and two (large) base tables. To speed up the delta evaluation, higher-order IVM materializes and recursively maintains update-independent parts of delta queries. This maintenance strategy creates a hierarchy of increasingly simpler materialized views that support each other's incremental maintenance.

Example 2 Consider the count query from Example 1 and the hierarchy of materialized views shown in Fig. 1. This IVM strategy materializes the top-level view M_Q along with four additional views M_R , M_S , M_T , and M_{ST} in order to speed up the maintenance task: for instance, updates to R and T exploit the precomputed partial counts in M_{ST} and respectively M_S and M_R for faster delta propagation. This approach is an instance of

Distributed Incremental View Maintenance, Fig. 1

View hierarchy for the query in Example 1. Delta propagations for updates to R (left blue) and to T (right red)



higher-order IVM since an update to one relation may cause the maintenance of several views. The triggers for updates to R and T look as:

```

ON UPDATE R BY ΔR:
  ΔMR := Sum[A](ΔR)
  ΔMQ := Sum[](ΔMR ⋙ MST)
  MR += ΔMR
  MQ += ΔMQ

ON UPDATE T BY ΔT:
  ΔMT := Sum[A,C](ΔT)
  ΔMST := Sum[A](MS ⋙ ΔMT)
  ΔMQ := Sum[](MR ⋙ ΔMST)
  MT += ΔMT
  MST += ΔMST
  MQ += ΔMQ

```

The trigger for updates to S is similar. For this example and this maintenance strategy, processing one update tuple takes a constant amount of work per statement: for instance, the join operation in the R trigger requires only one lookup in M_{ST} for each tuple of $ΔM_R$. Processing single-tuple updates in constant time is impossible to achieve in this example using classical incremental view maintenance, which evaluates $ΔQ = ΔR ⋙ S ⋙ T$, or recomputation, which evaluates $Q = (R ∪ ΔR) ⋙ S ⋙ T$. In both cases, the maintenance time is linear in the size of the database. □

View maintenance code, like that of Examples 1 and 2, consists of trigger statements, which are executed upon changes in the underlying database. The discussion so far has assumed that

the database and code are stored and respectively run on a single machine.

Distributed IVM

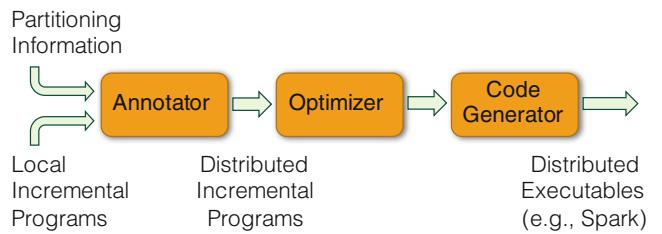
This section describes how to transform local maintenance code into maintenance programs optimized for running on large-scale processing platforms. The presented approach targets systems with synchronous execution, such as Spark (Zaharia et al. 2012) and Hadoop MapReduce (Dean and Ghemawat 2008), where one driver node orchestrates job execution among workers. Processing one batch of updates may require several computation stages, where each stage runs view maintenance code in parallel. All workers are stateful, preserve data between stages, and participate in every stage.

Materialized views are classified depending on the location of their contents. *Local views* are stored and maintained entirely on the driver node. They are suitable for materializing aggregates with small output domains. *Distributed views* have their contents spread over all workers to balance CPU and memory pressure. Each distributed view has an associated partitioning function that maps each tuple to a non-empty set of nodes storing its replicas.

Compilation Overview

Figure 2 shows the process of transforming a local incremental program into a functionally

Distributed Incremental View Maintenance, Fig. 2
Compilation of incremental programs



equivalent, distributed program. The input consists of maintenance statements expressed using the aforementioned query language. To distribute their execution, the compiler relies on the partitioning information about each materialized view, which is provided by the user.

The compilation process consists of three phases. The first phase annotates a given input program with partitioning information and, if necessary, introduces operators for exchanging data among distributed nodes in order to preserve the correctness of query evaluation. The second phase optimizes the annotated program using a set of simplification rules and heuristics that aim to minimize the number of jobs necessary for processing one update batch. The final stage uses code generation to specialize incremental programs for running on a specific processing platform.

Forming Distributed Programs

The semantics of the discussed query operators cannot be directly translated to distributed environments. For instance, unioning one local and one distributed view has no clear meaning. Even among distributed views only, naïvely executing query operators at each distributed node might yield wrong results. For instance, a natural join between two distributed views produces a correct result only if the views are identically partitioned over the join keys; otherwise, one or both operands need to be repartitioned.

The query language needs new location-aware primitives to allow reasoning about the correctness of distributed query evaluation. For that purpose, each query expression is extended (annotated) with a *location tag*: (1) Local tag

means the result is stored on the driver node, (2) $\text{Dist}(\mathcal{P})$ tag denotes the result is distributed among all workers according to partitioning function \mathcal{P} , and (3) Random tag means the result is randomly distributed among all workers. One relation or materialized view can take any of these tags. A query Q with a location tag T is written as Q^T .

To support distributed execution, the query language needs new operators for manipulating location tags and exchanging data over the network. These operators are called *location transformers*:

- Repart transformer reshuffles the distributed result of query Q using a partitioning function \mathcal{P} :

$$\text{Repart}_{\mathcal{P}}(Q^T) = Q^{\text{Dist}(\mathcal{P})}$$

where $T \in \{\text{Dist}(\cdot), \text{Random}\}$.

- Scatter transformer distributes the local result of query Q using a partitioning function \mathcal{P} :

$$\text{Scatter}_{\mathcal{P}}(Q^{\text{Local}}) = Q^{\text{Dist}(\mathcal{P})}$$

- Gather transformer aggregates the distributed result of query Q on the driver node:

$$\text{Gather}(Q^T) = Q^{\text{Local}}$$

where $T \in \{\text{Dist}(\cdot), \text{Random}\}$.

The location transformers are the only mechanism for exchanging data among nodes. Repart and Gather operate over distributed expres-

sions, while Scatter works with local expressions.

Location-Aware Query Operators. The semantics of the query operators is now extended to support location tags.

- Relation $R(A_1, A_2, \dots)^T$ stores its contents at location T.
- Bag union $Q_1^T + Q_2^T$ merges tuples locally or in parallel on every node, and the result retains tag T. Both operands must have the same location tag.
- Natural join $Q_1^T \bowtie Q_2^T$ has the usual semantics when $T = \text{Local}$. For distributed evaluation, both operands need to be partitioned on the join key; the result is also distributed and consists of locally evaluated joins on every node. Joins involving randomly partitioned operands are disallowed.
- Sum $_{A;f} Q^T$, when $T = \text{Dist}(\mathcal{P})$, computes partial aggregates on every node. The result retains tag T only if Q is key partitioned on one of the group by columns; otherwise, it gets a Random tag. When $T \neq \text{Dist}(\mathcal{P})$, the result retains tag T.

The remaining language constructs – constants, variable values, and comparisons – are location independent and can freely participate in both local and distributed expressions.

Algorithm. The procedure for transforming a local program into a valid distributed program operates as follows. For each program statement, the procedure starts by assigning a location tag to all relational terms in its expression tree based on the given partitioning scheme. Then, in a bottom-up fashion, it annotates each node of the tree with a location tag and introduces location transformers where necessary to preserve the semantics and correctness of each query operator, as discussed above. Upon reaching the root node of the statement, the procedure checks whether the right-hand side expression evaluates at the same location as the left-hand side view and, if not, introduces a Gather or Scatter transformer. The procedure repeats these steps for every statement.

Example 3 Consider the following statement obtained from the trigger for updates to T in Example 2.

$$M_{ST}(A) += \text{Sum}_{[A]}(M_S(A, B) \bowtie \Delta M_T(A, B))$$

D

Assume that M_{ST} and M_S are partitioned by A while ΔM_T is partitioned by B . Let $M^{[A]}$ denotes that M is partitioned on column A . The join operands M_S and ΔM_T have incompatible location tags, which requires a Repart transformer around one of the operands, say the left one:

$$\text{Repart}_{[B]}(M_S(A, B)^{[A]})^{[B]} \bowtie \Delta M_T(A, B)^{[B]}$$

The join result remains partitioned on B . The Sum expression computes partial aggregates grouped by column A . Such an expression cannot have location tag $[B]$ since B is not in the output schema; instead, it gets a Random tag. Upon reaching the root of the expression tree, another Repart ensures that the right-hand side expression has the same location tag as the target view.

$$\begin{aligned} M_{ST}(A)^{[A]} &+= \\ &\text{Repart}_{[A]}(\\ &\quad \text{Sum}_{[A]}(\text{Repart}_{[B]}(M_S(A, B)^{[A]})^{[B]} \\ &\quad \bowtie \Delta M_T(A, B)^{[B]})^{\text{Random}})^{[A]} \end{aligned}$$

□

The algorithm for constructing distributed expressions has no associated cost metrics and might produce suboptimal solutions. In the above example, the final statement involves two communication rounds (Reparts) between the driver and workers. But this overhead can be minimized: repartitioning the right join operator at first instead of the left one produces an equivalent, less expensive statement with only one communication round:

$$\begin{aligned} M_{ST}(A)^{[A]} &+= \\ &\text{Sum}_{[A]}(M_S(A, B)^{[A]} \bowtie \\ &\quad \text{Repart}_{[A]}(\Delta M_T(A, B)^{[B]})^{[A]})^{[A]} \end{aligned}$$

Optimizing distributed programs relies on a set of simplification and heuristic rules, which are presented next.

Optimizing IVM Programs

This section describes how to optimize distributed programs to minimize their processing and communication costs. The optimization procedures are divided into three groups based on their scope.

Local Optimization

Preprocessing a batch of updates can speed up incremental view maintenance. Early selection using a query's predicates eliminates irrelevant update tuples, while projection retains only the columns needed for the maintenance task. For instance, the R trigger of Example 2 preprocesses ΔR as $\text{Sum}_{[A]}(\Delta R(A, B))$ to retain only distinct A values needed for the later computation. Batch pre-aggregation can produce much fewer tuples and significantly decrease the maintenance cost and the amount of data sent over the network.

Intra-statement Optimization

Intra-statement optimizations aim to minimize the amount of network traffic generated by one distributed statement. The cost model considered here takes the number of communication rounds as the basic metric for cost comparison. Each location transformer (Repart, Scatter, and Gather) sends data over the network; thus, statements with fewer transformers require less communication. Considering more complex cost models that take into account the amount of data being exchanged is beyond the scope of this chapter.

The optimizer shown here also relies on few heuristics to resolve ties between statements with the same cost. For instance, it favors reshuffling of expressions that involve batch updates rather than whole materialized views as the former are usually smaller in size; also, it favors expressions with fewer Gather transformers to distribute the computation over workers as much as possible.

The optimizer uses a trial and error approach to recursively optimize a given statement. It tries to push each location transformer down the expression tree, following the rules from Fig. 3. Note that these rules might produce more expensive expressions, in which case the optimizer

$$\begin{aligned} \text{Repart}_{\mathcal{P}}(Q_1 \bowtie Q_2) &\Leftrightarrow \\ \text{Repart}_{\mathcal{P}}(Q_1) \bowtie \text{Repart}_{\mathcal{P}}(Q_2) & \\ \text{Repart}_{\mathcal{P}}(Q_1 + Q_2) &\Leftrightarrow \\ \text{Repart}_{\mathcal{P}}(Q_1) + \text{Repart}_{\mathcal{P}}(Q_2) & \\ \text{Repart}_{\mathcal{P}}(\text{Sum}_{A:f}(Q)) &\Leftrightarrow \\ \text{Sum}_{A:f}(\text{Repart}_{\mathcal{P}}(Q)) & \end{aligned}$$

Distributed Incremental View Maintenance, Fig. 3

Bidirectional optimization rules for Repart (same are for Scatter and Gather)

$$\begin{aligned} \text{Repart}_{\mathcal{P}}(Q^{\text{Dist}(\mathcal{P})}) &\Rightarrow Q^{\text{Dist}(\mathcal{P})} \\ \text{Gather}(Q^{\text{Local}}) &\Rightarrow Q^{\text{Local}} \\ \text{Repart}_{\mathcal{P}_1} \circ \text{Repart}_{\mathcal{P}_2} &\Rightarrow \text{Repart}_{\mathcal{P}_1} \\ \text{Repart}_{\mathcal{P}_1} \circ \text{Scatter}_{\mathcal{P}_2} &\Rightarrow \text{Scatter}_{\mathcal{P}_1} \\ \text{Gather} \circ \text{Repart}_{\mathcal{P}} &\Rightarrow \text{Gather} \\ \text{Gather} \circ \text{Scatter}_{\mathcal{P}} &\Rightarrow \text{Gather} \\ \text{Scatter}_{\mathcal{P}} \circ \text{Gather} &\Rightarrow \text{Repart}_{\mathcal{P}} \end{aligned}$$

Distributed Incremental View Maintenance, Fig. 4

Simplification rules for location transformers. The \circ sign is operator composition

backtracks. During each optimization step, it tries to simplify the current expression using the rules from Fig. 4. Each simplification rule always produces an equivalent expression with fewer location transformers.

In Example 3, the optimizer pushes the outer $\text{Repart}_{[A]}$ through the Sum and \bowtie operators and then simplifies $\text{Repart}_{[A]} \circ \text{Repart}_{[B]}$ to $\text{Repart}_{[A]}$. The optimized statement now requires only one communication round.

Inter-statement Optimization

Location transformers represent natural pipeline breakers in query evaluation since they need to materialize and shuffle the contents of their expression before continuing processing. Analyzing inter-statement dependencies can minimize the number of pipeline breakers and their communication overhead.

To facilitate inter-statement analysis, the optimizer first converts a given program into *single transformer form*, where each statement has at most one location transformer and that trans-

former, if present, always references one materialized view. This normalization procedure repeats the following three steps: (1) it materializes the contents being transformed (if not already materialized), (2) it extracts the location transformers targeting the materialized contents into separate statements, and (3) it updates the affected statements with new references. The optimizer normalizes each statement in a bottom-up traversal of its expression tree.

Example 4 Consider the maintenance trigger from Example 2 and assume that ΔR is randomly distributed among workers, M_R and M_{ST} are partitioned on A , and M_Q is stored at the driver node. The distributed trigger for updates to R consists of statements annotated with location tags in single transformer form:

```

ON UPDATE R BY ΔR:
1  ΔMRRandom := Sum[A](ΔRRandom)Random
2  ΔMR[A] := Repart[A](ΔMRRandom)[A]
3  ΔMQRandom := Sum[](ΔMR[A] ⋈ MST[A])Random
4  ΔMQLocal := Gather(ΔMQRandom)Local
5  MR[A] += ΔMR[A]
6  MQLocal += ΔMQLocal

```

Statement 1 pre-aggregates in parallel the update batch to minimize the amount of data being reshuffled by the subsequent statement. Statement 3 performs a distributed join followed by an aggregation that computes a partial count at each worker node. Statement 4 sums these partial counts at the driver. The remaining statements maintain the distributed view M_R and the local view M_Q . \square

The single transformer form sets clear boundaries around the contents that need to be communicated, which eases the implementation of further optimizations. Common subexpression elimination and dead code elimination can detect shared parts among statements and eliminate redundant network transfers. In contrast to the classical compiler optimizations, these routines take into account the location where each expression is executed.

The location tag of the expressions from both sides of one statement determines where and how that statement is executed. Statements targeting local materialized views are, as expected, executed at the driver in *local mode*. Statements involving distributed views run in *distributed mode*, where the driver initiates the computation. All location transformations run in *communication mode*, but materializing the contents to be reshuffled can happen in both local and distributed mode. For instance, preparing contents for Scatter takes place on the driver while for Repart and Gather happens on every worker.

Statement Blocks. Distributed statements are more expensive to execute than local statements. To run a distributed statement, the driver needs to serialize a task closure, ship it to all workers, and wait for the completion of each one of them. For short-running tasks, non-processing overheads can easily dominate in the execution time.

To amortize the cost of executing distributed statements, the optimizer can pack them together into processing units called *statement blocks*. A statement block consists of a sequence of distributed statements that can be executed at once on every node without compromising the program correctness. The optimizer analyzes the data-flow dependencies among statements to decide on valid statement reorderings.

Example 5 The trigger from Example 4 consists of three distributed statements (1, 3, and 5), two communication statements (2 and 4), and one local statement (6). Statements 3 and 5 can be grouped together into one block as the latter commutes with statement 4; thus, computing ΔM_Q^{Random} and updating $M_R^{[A]}$ happens in one distributed task. The same is not true for statements 1 and 3 because of the data-flow dependencies among the first three trigger statements. \square

To minimize network overheads, the optimizer can coalesce consecutive communication statements of the same type (i.e., with the same type of location transformer) into one compound request carrying a container with their individual payloads. This technique amortizes the intrinsic

start-up overhead associated with each network operation over multiple communication statements.

Conclusion

This chapter studies incremental view maintenance in local and distributed environments. The presented approach compiles local maintenance programs into distributed code optimized for the execution on large-scale processing platforms. Nikolic et al. (2016) provide details of the compilation process and analyze the performance of distributed IVM using Apache Spark as the data processing framework. The technique discussed here is general and applicable to other distributed data management systems and stream processing engines.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Apache Flink](#)
- ▶ [Continuous Queries](#)
- ▶ [Hadoop](#)
- ▶ [Incremental Approximate Computing](#)
- ▶ [Incremental Sliding Window Analytics](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Stream Processing Languages and Abstractions](#)
- ▶ [Robust Data Partitioning](#)
- ▶ [Spark SQL](#)
- ▶ [Stream Query Optimization](#)

References

- Chandramouli B, Goldstein J, Barnett M, DeLine R, Fisher D, Platt JC, Terwilliger JF, Wernsing J (2014) Trill: a high-performance incremental query processor for diverse analytics. *VLDB* 8(4):401–412
- Chirkova R, Yang J (2012) Materialized views. *Found Trends® Databases* 4(4):295–405
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Green TJ, Karvounarakis G, Tannen V (2007) Provenance semirings. In: *PODS*, pp 31–40
- Griffin T, Libkin L (1995) Incremental maintenance of views with duplicates. *SIGMOD Rec* 24(2):328–339

- Koch C (2010) Incremental query evaluation in a ring of databases. In: *PODS*, pp 87–98
- Koch C, Ahmad Y, Kennedy O, Nikolic M, Nötzli A, Lupei D, Shaikhha A (2014) DBToaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J* 23(2):253–278
- Murray DG, McSherry F, Isaacs R, Isard M, Barham P, Abadi M (2013) Naiad: a timely dataflow system. In: *SOSP*, pp 439–455
- Nikolic M, Dashti M, Koch C (2016) How to win a hot dog eating contest: distributed incremental view maintenance with batch updates. In: *SIGMOD*, pp 511–526
- Qian X, Wiederhold G (1991) Incremental recomputation of active relational expressions. *IEEE Trans Knowl and Data Eng* 3(3):337–341
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *NSDI*, pp 15–28
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: *SOSP*, pp 423–438

Distributed SPARQL Query Processing

- ▶ [Native Distributed RDF Systems](#)

Distributed Spatial Data Streaming

- ▶ [Streaming Big Spatial Data](#)

Distributed Systems

- ▶ [TARDiS: A Branch-and-Merge Approach to Weak Consistency](#)

Distributed View Update

- ▶ [Distributed Incremental View Maintenance](#)

Document Store

► NoSQL Database Systems

Dynamic Scaling

► Elasticity

Duplicate Detection

► Record Linkage

D

E

Elasticity

Vincenzo Gulisano¹, Marina Papatriantafilou¹,
and Alessandro V. Papadopoulos²

¹Chalmers University of Technology,
Gothenburg, Sweden

²Mälardalen University, Västerås, Sweden

Synonyms

Dynamic scaling; Provisioning and decommissioning

Overview

In data stream processing, elasticity refers to the ability of autonomously provisioning and decommissioning resources (e.g., threads or nodes) in order to match, at each point in time, the computational power needed to run a set of user-defined continuous queries.

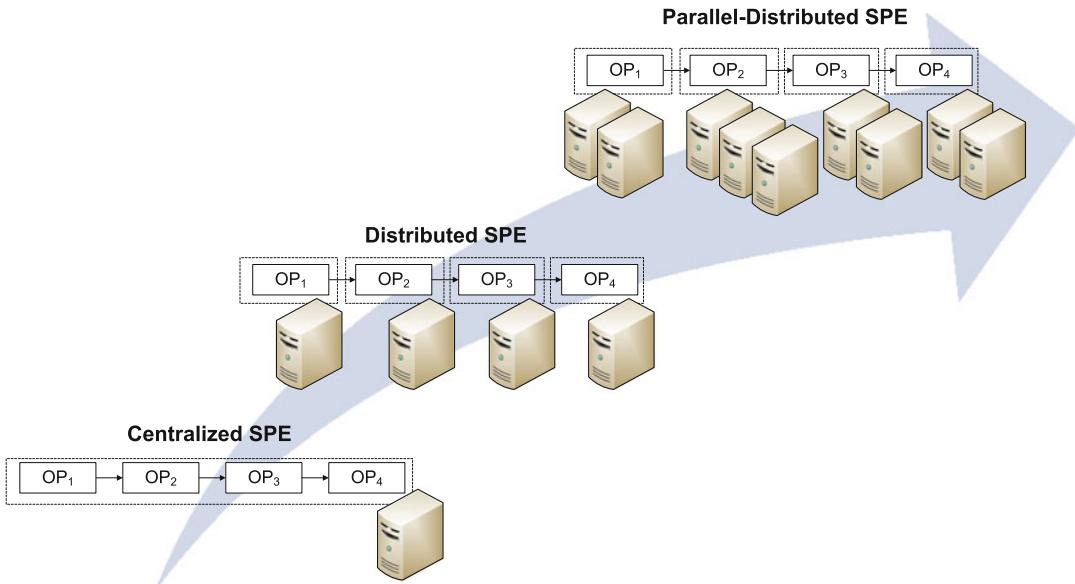
Elastic SPEs are able to dynamically scale up and out (or down and in, respectively) the resources employed to run the existing queries in order to match the needed computational power at each point in time. They can thus benefit of decoupled architectures such as cloud computing ones, and similar metrics for the performance evaluation can be defined (Ilyushkin et al. 2017). The variations in the needed computational power are due to the fluctuating nature

of data streams (both in terms of their volume and their underlying data distribution) and the variable number of queries added and removed over time by users.

Historical Background

Stream processing engines (SPEs) run user-defined continuous queries (or simply queries in the remainder). Each query is a directed acyclic graph of operators that consume and transform tuples coming in from a set of input sources and deliver output tuples to the user at the sink(s). Operators can be stateless, thus not maintaining a state that evolves based on the tuples being processed, or stateful, thus producing output tuples that depend on multiple input tuples. The distinction between stateless and stateful operators plays a significant role in the context of elasticity, as we elaborate in the following.

Since approximately the year 2000, SPEs evolved rapidly from centralized to parallel-distributed ones. As shown in Fig. 1, centralized SPEs, such as the pioneer SPE Aurora presented by Carney et al. (2002), deploy and run all the operators belonging to the same query at one node. Distributed SPEs such as the Borealis SPE, presented by Abadi et al. (2005), allow to run a query over multiple nodes, deploying each operator to exactly one node (i.e., providing inter-operator parallelism). Finally, parallel-distributed SPEs such as StreamCloud, introduced by Gulisano (2012); StreamMine3G, presented by



Elasticity, Fig. 1 Evolution from centralized SPEs to parallel-distributed SPEs, also showing how a sample query composed by four operators can be deployed at

Martin et al. (2014); Apache Spark (<https://spark.apache.org/>); Apache Flink (<https://flink.apache.org/>); or Heron (<https://twitter.github.io/heron/>) allow for individual operators to be run in parallel (i.e., providing intra-operator parallelism).

Before the introduction of elasticity protocols, the resources allocated to an SPE were statically assigned at the time the queries were deployed and their input streams were fed to them. In this case, the SPE running a certain set of queries could either be under-provisioned or over-provisioned.

When under-provisioned, the resources allocated to an SPE can cope with the average computational power needed to run the queries. In the occurrence of a temporary increase in the needed resources, nevertheless, the user will experience higher latency in the delivery of output tuples and might resort to adaptive techniques such as input admission control, as discussed by Balkesen et al. (2013), or load shedding, as discussed by Tatbul et al. (2003). On the other hand, the resources of an over-provisioned SPE are allocated to be able to cope with the highest peak of computational power that would be needed by the user-defined queries.

one or multiple nodes by the different SPEs. The figure is presented by Gulisano (2012) for the StreamCloud SPE

Under-provisioned SPEs are usually not an option when Service Level Agreements (SLAs) are in place, since they might incur monetary penalties. At the same time, over-provisioned SPEs result in wasted resources (and high maintenance costs).

It is important to notice that elasticity should (i) preserve the semantics of the analysis, i.e., not affect the correctness of a query's analysis, and (ii) require as little as possible involvement of the user in the decisions about resources to be provisioned or decommissioned. As we explain in the following, different solutions approach these aspects in different ways (and to different extents).

Foundations

We cover in this section elasticity's core aspects. As discussed by De Matteis and Mencagli (2017) with respect to SASO properties (i.e., stability, accuracy, settling time, and overshoot), elasticity mechanics build on top of contrasting objectives. On one hand, frequent adjustments of resources

to match variations in computational needs are desirable to overcome the limitations of over- or under-provisioned SPEs. At the same time, sporadic reconfigurations result in less overhead and temporary degradation of queries' performance.

For simplicity, we use the term *user* to refer to both who maintains the resources to run the SPE and who defines the queries running on the latter.

Provisioning and Decommissioning

Whenever the allocated resources should be adapted to match computational needs, elasticity can result in the provisioning of new threads or nodes (when more resources than the allocated ones are needed) or the decommissioning of some existing thread or node (when less resources than the allocated ones are needed). A first distinction in the existing elasticity protocols can be made between techniques that can provision (or decommission) one thread or node at a time and techniques that can provision (or decommission) multiple ones.

The solutions presented by Schneider et al. (2009) and Hochreiner et al. (2016), for instance, provide one-at-a-time provisioning and decommissioning of resources. The shortcoming of these techniques is that, in some cases, one extra thread or node might not be enough to match the increase of needed computational power. This is overcome by load-aware elastic techniques such as the ones discussed by Gulisano et al. (2012) and Heinze et al. (2014b). The latter decide how many resources should be provisioned or decommissioned based on the current computational load or the computational load estimated at the end of the elastic reconfiguration. By doing this, they reduce the reconfiguration overhead by encapsulating the provisioning or decommissioning of multiple threads or nodes rather than triggering multiple consecutive reconfigurations.

Load Balancing

Resource provisioning or decommissioning results in temporary performance degradation, while the deployment of queries (and their operators) over the allocated resources is undergoing a change (e.g., when waiting for

a new node to be initialized). Because of this, resources should be provisioned only when allocated ones (as a whole) cannot cope with the current workload. That is, provisioning (and decommissioning) of resources should not be triggered because of an uneven load distribution. This can be prevented by combining load balancing with elasticity.

Load balancing, nevertheless, usually boils down to NP-hard problems such as the bin-packing one, as discussed by Gulisano (2012), Balkesen et al. (2013), Heinze et al. (2013, 2014b), among others. When deciding the best workload repartitioning among allocated resources, it is also of key importance to leverage efficient state transfer techniques, as mentioned by Ding et al. (2015).

Granularity

Since centralized SPEs bound each query to exactly one node, they can only leverage techniques in which nodes are provisioned when a query, as a whole, cannot be run by the allocated ones, as proposed by Loesing et al. (2012).

Differently, distributed SPEs can also benefit of techniques in which operators' responsibility can be shifted between nodes, as proposed by Xing et al. (2005) and Heinze et al. (2013).

Finally, parallel-distributed SPEs can provision (or decommission) threads and nodes to adjust the parallelism degree of individual operators, as discussed by Gulisano et al. (2012), Heinze et al. (2014b), Martin et al. (2014), Castro Fernandez et al. (2013), among others.

Transferring of the responsibility for a whole operator (or part of a parallel one) to a new thread (or node) requires the rerouting of its input tuples to the new thread (or node). For stateful operators, furthermore, it also requires the transfer of its whole state (or part of it for parallel stateful operators).

Triggering Mechanism

A key aspect, besides the mechanics with which an elastic protocol assigns the analysis carried out by the resources before and after the reconfiguration, is the condition upon which such elastic reconfiguration should happen.

In the literature, the triggering conditions are usually based on (i) the threads' and nodes' CPU consumption, as discussed by Gulisano et al. (2012), Martin et al. (2014), Hochreiner et al. (2016), (ii) the operators' latency, as discussed by Zacheilas et al. (2015), De Matteis and Mencagli (2017), or (iii) the operators' queue lengths, as discussed by Kumbhare et al. (2015). An important distinction among the existing protocols can be made between reactive mechanisms and proactive ones. Reactive mechanisms, such as the one proposed by Gulisano et al. (2012), trigger a reconfiguration once a certain condition is met (e.g., when the CPU consumption exceeds a certain threshold). On the other hand, proactive mechanisms actively try to optimize the performance of the running queries (e.g., by constantly minimizing their latency), as for the solution discussed by De Matteis and Mencagli (2017), or predict (rather than observe) when a certain condition will be met, as discussed by Zacheilas et al. (2015).

Determinism

As we mentioned, an elastic reconfiguration can temporarily degrade the performance of an SPE while undergoing a reconfiguration (e.g., a temporary increase in the processing latency while waiting for a new node to be provisioned). Nevertheless, elasticity should not affect the correctness of the analysis. This property is known as determinism.

Determinism has been introduced in the context of parallel-distributed SPEs and in the context of algorithms for parallel data streaming operators, for instance, by Gulisano et al. (2016, 2017). The determinism guarantee states that the results produced by an operator run in parallel by multiple threads or at multiple nodes should produce the same exact results of its centralized counterpart. When all the parallel operators of a query run deterministically, then the query itself enjoys the determinism guarantee.

When elastically reconfiguring the resources allocated to a query, nevertheless, the system undergoes a transition during which the responsibility to process a certain subset of data and the

state currently maintained by stateful operators for such subset is shifted from one thread (or node) to another one. Enforcing determinism in this case implies that the results produced by a system while undergoing elastic reconfigurations are the exact same as the ones produced by a system whose resources are statically allocated. In practice, this means each tuple is processed exactly once, and the state is correctly preserved during each elastic reconfiguration, as we further explain when discussing state transfer.

Determinism is also referred to as *safety* by Hirzel et al. (2014) and *semantic transparency* by Gulisano et al. (2012).

State Transfer

When an elastic reconfiguration is triggered, the responsibility for the input tuples of an operator (a portion of them or all of them) is transferred from thread (or node) A to thread (or node) B . The main idea is to find a point in time P so that the processing of all tuples before P is A 's responsibility, while the processing of tuples coming after P is B 's responsibility.

This is trivial for stateless operators and provided by the protocols as the one proposed by Kumbhare et al. (2015). It is nevertheless challenging for stateful operator when enforcing determinism. The responsibility of the state maintained by A cannot be transferred to B before A is done processing all tuples up to P . A possible way of doing this guaranteeing determinism is by a blocking mechanism in which B cannot start processing tuples until it gets access to A 's state once the latter is done processing all tuples (Gulisano et al. 2012; Martin et al. 2014). When transferring the state by serializing it, sending it between two nodes, and finally deserializing it, nevertheless, the overhead and latency penalty of the reconfiguration can degrade the SPE performance. This can be alleviated by techniques aiming at reducing latency spikes, such as the ones proposed by Heinze et al. (2014a), or at recreating small states at B by sending to the latter previously sent tuples (rather than transferring A 's state).

Syntactic Transparency

Together with determinism, syntactic transparency is another desirable feature to make elastic reconfigurations oblivious to the user. When an SPE is syntactically transparent, the user does not need to trigger reconfigurations manually but can rely on the SPE itself to monitor the resource utilization and adjust the latter depending on the variations of the former. The minimum information provided by the user is the pool from which resources can be provisioned and to which resources can be decommissioned over time.

Further information might be needed depending on the operators deployed by the user. When using the standard operators provided by SPEs, their elastic reconfiguration protocols can handle provisioning and decommissioning without support from the user, as discussed by Gulisano et al. (2012). When relying on user-defined operators, on the other hand, an SPE can provide a general interface to serialize and deserialize the state of stateful operators when a reconfiguration is triggered, as discussed by Martin et al. (2014). Alternatively, an SPE could opt for maintaining operators' state decoupled from the individual operators themselves, exposing it with shared state management primitives, as proposed by Castro Fernandez et al. (2013).

Examples of Applications

In the literature, elasticity has been discussed in relation to many real-world applications. All these applications share data sources with fluctuating input rates that, once fed to the queries run by an SPE, need varying amounts of resources over time in order to cope with the analysis' computational costs.

Gulisano (2012) evaluated the elasticity protocols of the StreamCloud SPE for fraud detection applications (both for credit card transactions and mobile phone networks). Martin et al. (2014) focused on elasticity for clickstream analysis while, Zacheilas et al. (2015) studied the benefit of elasticity in the context of transportation and mobility applications (more specifically, for traf-

fic monitoring of buses in the city of Dublin). Transportation is also discussed by Hochreiner et al. (2016), with a focus on taxi rides, and by Kumbhare et al. (2015), evaluating their solution with the Linear Road benchmark. The benchmark simulates traffic for a set of highways, and the queries are designed to notify drivers of both their variable tolls and of existing accidents, as discussed by Arasu et al. (2004).

Elasticity has also been discussed for applications analyzing social media such as Twitter, by Ding et al. (2015), and applications running financial market analysis, by Gedik et al. (2014), Heinze et al. (2014b), and De Matteis and Mencagli (2017).

Future Directions for Research

Elasticity has been studied extensively in the literature, resulting in many protocols that can cover the presented core aspects to different extents.

New challenges in large distributed architectures such as cyber-physical systems and fog architectures are given by their heterogeneous nature (both in terms of resources and applications). Successful use of elastic protocols will require to properly leverage the computational power of each individual node, that is, to properly scale up before scaling out, as discussed by Gibbons (2015). This can be enabled by elastic protocols that can properly leverage nonhomogeneous hardware, as discussed by Koliousis et al. (2016), and fine-grained synchronization, as discussed by Gulisano et al. (2016, 2017). At the same time, it will require elastic protocols to be application-aware, since SLAs and physical constraints external to SPEs themselves (e.g., the latency incurred when communicating with a certain node) will both affect the elastic reconfigurations triggered by an SPE.

Cross-References

- ▶ [Apache Flink](#)
- ▶ [Apache Spark](#)

- ▶ [Cloud Computing for Big Data Analysis](#)
- ▶ [Continuous Queries](#)
- ▶ [Sliding-Window Aggregation Algorithms](#)
- ▶ [Stream Window Aggregation Semantics and Optimization](#)
- ▶ [StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing](#)

References

- Abadi DJ, Ahmad Y, Balazinska M, Cetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey A, Rasin A, Ryvkina E et al (2005) The design of the borialis stream processing engine. In: CIDR, vol 5, pp 277–289
- Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Proceedings of the thirtieth international conference on very large data bases, VLDB Endowment, vol 30, pp 480–491
- Balkesen C, Tatbul N, Özsu MT (2013) Adaptive input admission and management for parallel stream processing. In: Proceedings of the 7th ACM international conference on distributed event-based systems. ACM, pp 15–26
- Carney D, Çetintemel U, Cherniack M, Convey C, Lee S, Seidman G, Stonebraker M, Tatbul N, Zdonik S (2002) Monitoring streams: a new class of data management applications. In: Proceedings of the 28th international conference on very large data bases, VLDB Endowment, pp 215–226
- Castro Fernandez R, Migliavacca M, Kalyvianaki E, Pietzuch P (2013) Integrating scale out and fault tolerance in stream processing using operator state management. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. ACM, pp 725–736
- De Matteis T, Mencagli G (2017) Proactive elasticity and energy awareness in data stream processing. *J Syst Softw* 127:302–319
- Ding J, Fu TZJ, Ma RTB, Winslett M, Yang Y, Zhang Z, Chao H (2015) Optimal operator state migration for elastic data stream processing. *CoRR* abs/1501.03619
- Gedik B, Schneider S, Hirzel M, Wu KL (2014) Elastic scaling for data stream processing. *IEEE Trans Parallel Distributed Syst* 25(6):1447–1463
- Gibbons PB (2015) Big data: scale down, scale up, scale out. In: IPDPS, p 3
- Gulisano V (2012) Streamcloud: an elastic parallel-distributed stream processing engine. PhD thesis, Universidad Politécnica de Madrid
- Gulisano V, Jimenez-Peris R, Patino-Martinez M, Soriente C, Valduriez P (2012) Streamcloud: an elastic and scalable data streaming system. *IEEE Trans Parallel Distrib Syst* 23(12):2351–2365
- Gulisano V, Nikolakopoulos Y, Papatriantafilou M, Tsigas P (2016) Scalejoin: a deterministic, disjoint-parallel and skew-resilient stream join. *IEEE Trans Big Data* PP(99):1–1
- Gulisano V, Nikolakopoulos Y, Cederman D, Papatriantafilou M, Tsigas P (2017) Efficient data streaming multiway aggregation through concurrent algorithmic designs and new abstract data types. *ACM Trans Parallel Comput* 4(2):11:1–11:28
- Heinze T, Ji Y, Pan Y, Grueneberger FJ, Jerzak Z, Fetzer C (2013) Elastic complex event processing under varying query load. In: BD3@ VLDB, pp 25–30
- Heinze T, Jerzak Z, Hackenbroich G, Fetzer C (2014a) Latency-aware elastic scaling for distributed data stream processing systems. In: Proceedings of the 8th ACM international conference on distributed event-based systems. ACM, pp 13–22
- Heinze T, Pappalardo V, Jerzak Z, Fetzer C (2014b) Auto-scaling techniques for elastic data stream processing. In: IEEE 30th international conference on data engineering workshops (ICDEW). IEEE, pp 296–302
- Hirzel M, Soulé R, Schneider S, Gedik B, Grimm R (2014) A catalog of stream processing optimizations. *ACM Comput Surv* 46(4):46:1–46:34
- Hochreiner C, Vögler M, Schulte S, Dustdar S (2016) Elastic stream processing for the internet of things. In: IEEE 9th international conference on cloud computing (CLOUD). IEEE, pp 100–107
- Ilyushkin A, Ali-Eldin A, Herbst N, Papadopoulos AV, Ghit B, Epema D, Iosup A (2017) An experimental performance evaluation of autoscaling algorithms for complex workflows. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering (ICPE), ICPE'17. ACM, New York, pp 75–86. <http://doi.org/10.1145/3030207.3030214>
- Koliosius A, Weidlich M, Castro Fernandez R, Wolf AL, Costa P, Pietzuch P (2016) Saber: window-based hybrid stream processing for heterogeneous architectures. In: Proceedings of the 2016 international conference on management of data. ACM, pp 555–569
- Kumbhare AG, Simmhan Y, Frincu M, Prasanna VK (2015) Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. *IEEE Trans Cloud Comput* 3(2):105–118
- Loesing S, Hentschel M, Kraska T, Kossmann D (2012) Stormy: an elastic and highly available streaming service in the cloud. In: Proceedings of the 2012 joint EDBT/ICDT workshops. ACM, pp 55–60
- Martin A, Brito A, Fetzer C (2014) Scalable and elastic realtime click stream analysis using streammine3g. In: Proceedings of the 8th ACM international conference on distributed event-based systems. ACM, pp 198–205
- Schneider S, Andrade H, Gedik B, Biem A, Wu KL (2009) Elastic scaling of data parallel operators in stream processing. In: IEEE international symposium on parallel & distributed processing, IPDPS 2009. IEEE, pp 1–12
- Tatbul N, Çetintemel U, Zdonik S, Cherniack M, Stonebraker M (2003) Load shedding in a data stream manager. In: Proceedings of the 29th international conference on very large data bases, VLDB Endowment, vol 29, pp 103–114

- ference on very large data bases, VLDB Endowment, vol 29, pp 309–320
- Xing Y, Zdonik S, Hwang JH (2005) Dynamic load distribution in the borealis stream processor. In: Proceedings 21st international conference on data engineering, ICDE 2005. IEEE, pp 791–802
- Zacheilas N, Kalogeraki V, Zygouras N, Panagiotou N, Gunopoulos D (2015) Elastic complex event processing exploiting prediction. In: IEEE international conference on big data (Big Data). IEEE, pp 213–222

Electronic Health Data (EHD)

- ▶ [Security and Privacy in Big Data Environment](#)

Electronic Health Records (EHR)

- ▶ [Security and Privacy in Big Data Environment](#)

ELT

- ▶ [ETL](#)

Emerging Hardware Technologies

Xuntao Cheng¹, Cheng Liu², and Bingsheng He²

¹Nanyang Technological University, Jurong West, Singapore

²National University of Singapore, Singapore, Singapore

Definitions

This chapter introduces emerging hardware technologies such as GPUs, x86-based many-core processors, and die-stacked DRAMs, which have significant impacts on big data applications.

Overview

This chapter introduces emerging hardware technologies and their impacts on big data applications. For processors, this chapter includes GPUs, Intel Xeon Phi many-core processors, FPGAs, and specialized processors. This chapter also covers many new memory technologies such as die-stacked DRAMs, storage class memory, etc. At last, we introduce network interconnects.

E

Outline

We introduce emerging hardware including processors, memory, hard disk, and network. On the processors, we focus on many-core processors (GPUs and Intel Xeon Phi), FPGA, and specialized hardware.

Many-Core Processors

GPU

Graphics processing units (GPUs) are specialized processors initially designed to alter memory at a massive scale and fast speed to accelerate the creation of images. They have been redesigned to support programmable shading in 2001 and gradually become general purpose accelerators. A GPU dedicates most of its transistors to simpler and smaller cores with more registers to support massive thread-level parallelism on a large scale of data. The bandwidth of GPU's memory is usually higher than that of the main memory. These features benefit a wide range of big data applications such as business intelligence and data mining (Christoph Weyerhaeuser et al. 2008; Ren Wu et al. 2009).

NVIDIA has proposed the NVLink technology, a high bandwidth, energy-efficient interconnect that enables ultrafast communications between the CPU and the GPU (NVIDIA 2017). The bandwidth of NVLink is 5–12 times faster than that of the PCIe Gen 3. NVLink can also connect multiple GPU cards in a mesh, improving GPUs' capabilities for big data analytics.

Intel Xeon Phi

Intel introduced two generations of x86-based many-core processors branded as Intel Xeon Phi in 2012 and 2016, respectively (Intel 2017f). Xeon Phi tries to offer a general purpose x86-based processor with a high performance that is competitive with tailored accelerators. Important features of the latest Xeon Phi processor include 512-bit vector processing units supporting the AVX-512 ISA (Intel 2017d), die-stacked High Bandwidth Memory (HBM), and up to x86-based 72 cores. Xeon Phi is also binary compatible with Xeon processors, enabling an easy transition for big data applications from multi-core processors to many-core processors.

Intel proposed its Omni-Path Architecture, a high bandwidth low-latency interconnect featuring multiple flow and routing optimizations and aiming at scaling processing units to tens of thousands of cores at a competitive price to support future HPC workload (Intel 2017e). Similar to NVLink, Omni-Path can connect multiple Xeon and Xeon Phi processors to facilitate communications among processors.

FPGA

Field-programmable gate array (FPGA) is an integrated circuit which consists of an array of configurable logic blocks (CLBs) connected via programmable interconnects. It can be reprogrammed and provide application-specific customization for target application. The use of FPGA has been demonstrated to be successful in many applications. This motivates the cloud vendors to integrate FPGAs in the cloud and provide FPGAs as a service (Amazon 2017; Aliyun 2017).

FPGA evolves rapidly over the years, and here are the major features that are worth for highlighting:

- Hybrid system architecture: closely coupled CPU-FPGA architecture such as Intel Xeon-FPGA (Herman and Randy 2016) emerges recently. CPU and FPGA have a shared mem-

ory and coherent cache support facilitating efficient hardware/software co-design.

- Large on-chip resources: By using the latest semiconductor technologies such as 16Fin FET+ process (Xilinx 2017b), stacked silicon interconnect (Xilinx 2017a), and Intel 14 nm Tri-Gate process (Intel 2017c), new generation of FPGAs has much more resources especially DSP slices and on-chip RAM blocks up to 500Mb and allows high-frequency hardware design up to 1 GHz (Intel 2017a). The large amount of on-chip resources exhibit the great potential for compute-intensive and data-intensive applications.
- High bandwidth external memory: High bandwidth memory up to 460GB/s such as HBM (Xilinx 2017c) and HMC (Intel 2017b) is integrated with the FPGAs.
- High performance networking: 100 Gigabit Ethernet or 400 Gigabit Ethernet is provided for high performance interconnection (Intel 2017g).

Specialized Processors

Specialized processors have long been a viable option for conventional processors. Recently, due to the emergence of machine learning, specialized processor is an application-specific integrated circuit (ASIC) developed specifically for neural network and deep learning. Many different specialized processors have been fabricated in the past few years. Google has built TPU for the NN applications in its data center. The first generation of TPU targets NN inference only (Norman et al. 2017). It achieves both high throughput and low latency for inference. The second generation of TPU supports both inference and training (Geoffrey 2017). Huawei has NPU embedded in their mobile phones for faster and more energy-efficient machine learning (Anandtech 2017).

Memory

Die-Stacked DRAM

Die-stacked DRAM is an emerging type of memory that is integrated closely to processing units in

the same package. By stacking multiple layers of DRAMs together in the 3D space, the bandwidth per stack can be increased a lot. For example, the third generation of the High Bandwidth Memory (HBM), a JEDEC standard of die-stacked DRAM, is expected to reach a peak bandwidth of 512 GB/s per module in the near future (JEDEC Solid State Technology Association 2014, 2015). Die-stacked DRAMs can be either interposed by the side of the processor or placed on top of the processor. Although both approaches can achieve a very high bandwidth, the later approach also reduces the memory access latency.

Storage Class Memory

Storage class memory (SCM) is a type of byte-addressable nonvolatile memory that has similar performance with DRAM in terms of latency and similar capacity and economics with the storage (Burr et al. 2008). It is an attractive solution to meet the needs for fast and scalable memory in big data analytics, databases, cloud services, and many other scenarios. By far, candidates for SCM include phase-change memory (PCM), spin-transfer torque memory (STT-RAM), Intel 3D Xpoint, resistive RAM, HP memristors, Samsung V-NAND, and others. SCM has asynchronous read and write latency. For example, the read and write latencies of STT-RAM are 10 ns and 50 ns, respectively. This calls for careful software optimizations.

PIM

Processing in memory (PIM) is a technique that integrates the processing logic with the memory so that the tasks that don't require extensive processing can be done directly within the memory. It avoids the frequent data transfers between the processing units and memory in conventional computing architectures. Thus, it helps to speed up processing, memory transfer rate, and memory bandwidth and decrease processing latency. Centring the PIM concept, a number of PIM architectures have been explored over the years. Some of the researchers proposed to integrate the processing logic with DRAM by leveraging the 3D technologies (Qiuling Zhu et al. 2013). Some of the researchers opted to build processing logic

together with the memory cells based on different memory techniques such as metal-oxide resistive random access memory (ReRAM) (Wong et al. 2012), spin-transfer torque magnetic RAM (STT-RAM) (Adrien et al. 2015), and phase-change memory (PCM) (Geoffrey et al. 2015).

E

Hard Disk

There are recent efforts on improving the hard disk drives (HDD), whose applications have been threatened by its slower bandwidth compared with other emerging memory technologies. A new approach to improve HDDs tries to replace the conventional aluminum substrates with a new prototyping glass substrates. Glass substrates can help in increasing the storage space of a single HDD to about 20 TB (Jon Martindale 2017). This enlarged capacity may improve the storage capability of disks serving big data systems.

Network

Smart NIC

Smart network interface cards (NIC) are NICs equipped with processors to process data while transferring them in the network. These processors can be FPGAs, CPUs, or specifically designed ASICs. Although some computation tasks can be offloaded to conventional NICs, they are usually not complex, such as calculating the checksum. Smart NICs can exploit its processors to support much more complex tasks such as the network functions virtualization (NFV) and more complicated data processing tasks.

RDMA

Remote direct memory access (RDMA) over Converged Ethernet (RoCE) is originally used by the InfiniBand interconnect technology and now applied on Ethernet networks. It allows offloading network transfers from the CPU to RDMA hardware engines, which frees the CPUs for other tasks which in turn increases the performance of tasks. This feature has found very beneficial for big data systems (Lu et al. 2014).

Cross-References

- ▶ Big Data and Exascale Computing
- ▶ Big Data Architectures
- ▶ Computer Architecture for Big Data
- ▶ Distributed File Systems
- ▶ GPU-Based Hardware Platforms
- ▶ Network-Level Support for Big Data Computing
- ▶ Parallel Processing with Big Data

References

- Aliyun (2017) FPGA cloud server. <https://cn.aliyun.com/product/ecs/fpga>. Online; Accessed 11 Nov 2017
- Amazon (2017) Amazon EC2 F1 instances. https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls. Online; Accessed 11 Nov 2017
- Anandtech (2017) Huawei shows unannounced Kirin 970 at IFA 2017: dedicated neural processing unit. <https://soylentnews.org/article.pl?sid=17/09/06/1127209>. Online; Accessed 11 Nov 2017
- Burr GW, Kurdi BN, Scott JC, Lam CH, Gopalakrishnan K, Shenoy RS (2008) Overview of candidate device technologies for storage-class memory. *IBM J Res Dev* 52(4.5):449–464
- Burr GW, Shelby RM, Sidler S, Di Nolfo C, Jang J, Boybat I, Shenoy RS, Narayanan P, Virwani K, Giacometti EU et al (2015) Experimental demonstration and tolerancing of a large-scale neural network (1,65,000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans Electron Devices* 62(11):3498–3507
- Google (2017) Google TPU. <https://cloud.google.com/tpu/>. Online; Accessed 26 Mar 2018
- Intel (2017a) A new FPGA architecture and leading-edge FinFET process technology promise to meet next-generation system requirements. https://www.altera.com/en_US/pdfs/literature/wp/wp-01220-hyperflex-architecture-fpga-socs.pdf. Online; Accessed 11 Nov 2017
- Intel (2017b) Hybrid memory cubes. <https://www.altera.com/solutions/technology/serial-memory/hybrid-memory-cubes.html>. Online; Accessed 11 Nov 2017
- Intel (2017c) Intel 14 nm technology. <https://www.intel.sg/content/www/xa/en/silicon-innovations/intel-14nm-technology.html>. Online; Accessed 11 Nov 2017
- Intel (2017d) Intel advanced vector extensions 512. <https://www.intel.sg/content/www/xa/en/architecture-and-technology/avx-512-overview.html>. Online; Accessed 12 Nov 2017
- Intel (2017e) Intel® Omni-Path Fabric 100 series. <https://www.intel.sg/content/www/xa/en/high-performance-computing-fabrics/omni-path-architecture-fabric-overview.html>. Online; Accessed 12 Nov 2017
- Intel (2017f) Intel Xeon Phi processors. <https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html>. Online; Accessed 12 Nov 2017
- Intel (2017g) Transceiver technology. <https://www.altera.com/solutions/technology/transceiver/overview.html>. Online; Accessed 11 Nov 2017
- JEDEC Solid State Technology Association (2014) Wide I/O single data rate (Wide I/O SDR), JESD229. <https://www.jedec.org/system/files/docs/JESD229.pdf>
- JEDEC Solid State Technology Association (2015) High bandwidth memory (HBM) DRAM, JESD235A. <https://www.jedec.org/system/files/docs/JESD235A.pdf>
- Jouppi NP, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Al Borchers et al (2017) In-datacenter performance analysis of a tensor processing unit. *arXiv preprint. arXiv:1704.04760*
- Lu X, Rahman MWU, Islam N, Shankar D, Panda DK (2014) Accelerating spark with RDMA for big data processing: early experiences. In: 2014 IEEE 22nd annual symposium on high-performance interconnects, pp 9–16
- Martindale J (2017) Hard drives of the future will be faster and larger thanks to new glass platters. <https://www.digitaltrends.com/>. Online; Accessed 26 Dec 2017
- NVIDIA (2014) NVIDIA NVLink TM high-speed interconnect. <http://info.nvidianews.com/rs/nvidia/images/NVIDIA%20NVLink%20High-Speed%20Interconnect%20Application%20Performance%20Brief.pdf>. Online; Accessed 12 Nov 2017
- Schmit H, Huang R (2016) Dissecting xeon+ FPGA: why the integration of cpus and FPGAs makes a power difference for the datacenter. In: Proceedings of the 2016 international symposium on low power electronics and design. ACM, pp 152–153
- Vincent AF, Larroque J, Locatelli N, Romdhane NB, Bichler O, Gamrat C, Zhao WS, Klein J-O, Galdin-Retailleau S, Querlioz D (2015) Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *IEEE Trans Biomed Circuits Syst* 9(2):166–174
- Weyerhaeuser C, Mindnich T, Faerber F, Lehner W (2008) Exploiting graphic card processor technology to accelerate data mining queries in sap netweaver bia. In: IEEE international conference on data mining workshops, 2008 (ICDMW'08). IEEE, pp 506–515
- Wong H-SP, Lee H-Y, Yu S, Chen Y-S, Wu Y, Chen P-S, Lee B, Chen FT, Tsai M-J (2012) Metal-oxide RRAM. *Proc IEEE* 100(6):1951–1970
- Wu R, Zhang B, Hsu M (2009) Gpu-accelerated large scale analytics. *IACM UCHPC*
- Xilinx (2017a) All programmable 3D ICs. <https://www.xilinx.com/products/silicon-devices/3dic.html>. Online; Accessed 11 Nov 2017
- Xilinx (2017b) Delivering a generation ahead at 20 nm and 16 nm. <https://www.xilinx.com/about/generation-ahead-16nm.html>. Online; Accessed 11 Nov 2017.

Xilinx (2017c) Integrated HBM and RAM. <https://www.xilinx.com/products/technology/memory.html>. Online; Accessed 11 Nov 2017

Zhu Q, Akin B, Sumbul HE, Sadi F, Hoe JC, Pileggi L, Franchetti F (2013) A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing. In: 3D systems integration conference (3DIC), 2013 IEEE international. IEEE, pp 1–7

End-to-End Benchmark

Milind A. Bhandarkar
Ampool, Inc., Santa Clara, CA, USA

Synonyms

Deep analytics benchmark; Pipeline benchmark

Definitions

An end-to-end data pipeline benchmark is a standardized suite of data ingestion, data processing, and data queries, arranged in a series of stages, where the output of a previous stage in a pipeline feeds the next stage in pipeline, exercising all the needed system characteristics for commonly constructed data pipeline workloads.

Historical Background

As we witness the rapid transformation in data architecture, where relational database management systems (RDBMS) are being supplemented by large-scale non-relational stores such as Hadoop Distributed File System (HDFS), MongoDB, Apache Cassandra, and Apache HBase, a more fundamental shift is on its way, which would require larger changes to modern data architectures. While the current shift was mandated by business requirements for the connected world, the next wave will be dictated by operational cost optimization, transformative changes in the

underlying infrastructure technology, and newer use-cases, such as Internet of Things (IoT), deep learning, and conversational user interfaces (CUI).

These new use-cases combined with ubiquitous data characterized by its large volume, variety, and velocity have resulted in an emergence of big data phenomenon. Most applications being built in this connected world are based on analyzing historical data of various genres to create a context within which the user interaction with applications is interpreted. Such applications are popularly referred to as data-driven applications. While such data-driven applications are coming into existence across almost all the verticals, such as financial services, healthcare, manufacturing, and transportation, web-based and mobile advertising companies recognized the potential of big data technologies for building data-driven applications and were arguably the earliest adopters of big data technologies. Indeed, popular big data platforms such as Apache Hadoop were created and adopted at web technology companies, such as Yahoo, Facebook, Twitter, and LinkedIn.

Most of the initial uses of these big data technologies were in workloads analyzing large unstructured and semi-structured datasets in batch-oriented Java MapReduce programs or using queries in SQL-like declarative languages. However, availability of the majority of enterprise data on big data distributed storage infrastructure soon attracted other non-batch workloads to these platforms. Massively parallel SQL engines such as Apache Impala and Apache HAWQ that provide interactive analytical query capabilities were developed to natively run on top of Hadoop analyzing data stored in Hadoop Distributed File System (HDFS). Since 2012, cluster resource management, scheduling, and monitoring functionality previously embedded in MapReduce framework have been modularly separated out as a separate component, Apache YARN, in the Hadoop ecosystem. This allowed data processing frameworks other than MapReduce to natively run on Hadoop clusters, sharing computational resources among various frameworks. The last 4 years have seen an explosion of Hadoop-native data processing frameworks

tackling various processing paradigms, such as streaming analytics, OLAP cubing, iterative machine learning, and graph processing, among others.

Flexibility of HDFS (and other Hadoop-compatible storage systems), sometimes federated across a single computational cluster for storing large amounts of structured, semi-structured, and unstructured data, and the added flexibility and choice of data processing frameworks, has resulted in Hadoop-based big data platforms that are being increasingly utilized for end-to-end data pipelines.

Emergence of this ubiquitous new platform architecture has created the imperative for an industry standard benchmark suite that evaluates efficacy of such platforms for end-to-end data processing workloads for consumers to make informed choices about various storage infrastructure components and computational engines.

In this article, we describe an *end-to-end data pipeline* benchmark, based on real-life workloads. While the benchmark itself is based on a typical pipeline in online advertising, it shares many common patterns across a variety of industries that make it relevant to a majority of big data deployments. In the next section, we discuss various previous benchmarking efforts that have contributed to our current understanding of industry standard big data benchmarks. Then we describe the benchmark scenario, at a fictitious Ad-Tech company. The datasets stored and manipulated in the end-to-end data pipeline of this benchmark are discussed along with various computations that are performed on the data in data pipeline. We propose metrics to measure the effectiveness of systems under test (SUT), report based on various classes of benchmarks that correspond to data sizes, and conclude by outlining future work.

Related Work

Performance benchmarking of data processing systems is a rich field of study and has seen tremendous activity since commercial

data platforms were first developed several decades ago. Several industry consortia have been established to standardize benchmarks for data platforms over the years. Of these, Transaction Processing Performance Council (TPC) and Standard Performance Evaluation Corporation (SPEC), both established in 1988, have gained prominence and have created several standard benchmark suites. While TPC has traditionally been focused on data platforms, SPEC has produced several microbenchmarks aimed at evaluating low-level computer systems performance. Among the various TPC standard benchmarks, TPC-C (for OLTP workloads), TPC-H (for data warehousing workloads), and TPC-DS (for analytical decision support workloads) have become extremely popular, and several results have been published using these benchmark suites. The benchmark specifications consist of data models, tens of individual queries on these data, synthetic data generation utilities, and a test harness to run these benchmarks and to measure performance.

Starting late 2011, the Center for Large-Scale Data Systems Research (CLDS) at the San Diego Supercomputing Center, University of California at San Diego, along with several industry and academic partners, established a *Big Data Benchmarking Community (BDBC)* to establish standard benchmarks in evaluating hardware and software systems that form the modern big data platforms. A series of well-attended workshops called *Workshop on Big Data Benchmarking* have been conducted across three continents as part of this effort, and proceedings of those workshops have been published.

Recognizing the need to speed up development of industry standard benchmarks, especially for rapidly evolving big data platforms and workloads, a new approach to developing and adopting benchmarks within the TPC, called *TPC-express*, was adopted in 2014. Since then, two standard benchmarks, TPC-xHS and TPC-xBB, have been created under TPC-express umbrella. TPC-xHS provides standard benchmarking rigor to the existing Hadoop Sort (based on Terasort) benchmark, whereas TPC-xBB provides big data extensions to the data models and workloads

from the earlier TPC-DS benchmarks. TPC-xHS is a microbenchmark, which evaluates the shuffle performance in MapReduce and other large data processing frameworks. TPCx-BB is based on a benchmark proposal named *BigBench* (Ghazal et al. 2013) that consists of a set of 30 modern data warehousing queries that combine both structured and unstructured data. None of the existing benchmarks provide a way to evaluate combined performance of an end-to-end data processing pipelines.

The need for having an industry standard big data benchmark for evaluating performance of data pipelines and conducting an annual competition to rank the top 100 systems based on this benchmark has been described in (Baru et al. 2013). AdBench, the benchmark described in this article, builds upon this deep analytics pipeline benchmark proposal, by extending the proposed user-behavior deep analytics pipeline, to include streaming analytics and ad hoc analytical queries, in the presence of transactions on the underlying datasets.

AdBench represents a generational advance over the previous standardized benchmarks, because it combines various common computational needs into a single end-to-end data pipeline. We note that while the performance of individual tasks in data pipelines remains important, operational complexity of assembling an entire pipeline composed of many different systems adds overheads of data exchange between various computational frameworks. Data layouts optimized for different tasks require transformations. Scheduling different tasks into a single data pipeline requires a workflow engine with its own overheads. These additional data pipeline composition and orchestration are not taken into account by the previous benchmarks that measure effectiveness of systems on individual tasks. We believe that systems that exhibit good-enough performance to implement entire data pipelines are preferred for their operational simplicity, rather than best-of-breed disparate but incompatible systems that may exhibit the best performance for individual tasks, but have to be integrated with external glue code. Thus, our data pipeline benchmarks measure

performance of an end-to-end data pipeline, rather than focus on *hero numbers* for individual tasks.

E

Foundations

Acme is a very popular content aggregation company that has a web-based portal and also a mobile app, with tens of millions of users, who frequently visit them from multiple devices several times a day to get hyper-personalized content, such as news, photos, audio, and video. Acme has several advertising customers (we distinguish between *users*, who browse through Acme's website, from *customers*, who publish advertisements on that website) who pay them to display their advertisements on all devices. Acme is one of the many Web 3.0 companies because they have a deep understanding of their users' precise interests as well as exact demographic data, history of consumption of content, and interactions with advertisements displayed on Acme website and mobile application. Acme personalizes their users' experience based on this data. They have an ever-growing taxonomy of their users' interests, and their advertisers can target users not only by the demographics but also by the precise interests of those users.

Here is Acme's business in numbers:

- 100 million registered users, with 50 million daily unique users
- 100,000 advertisements across 10,000 advertisements campaigns
- 10 million pieces of content (news, photos, audio, video)
- 50,000 keywords in 50 topics and 500 subtopics as user interests, content topics, and for ad targeting

Acme has several hundred machines serving advertisements, using a unique matching algorithm that fetches a user's interests, finds the best match within a few milliseconds, and serves the ad within appropriate content.

Acme has many data scientists and Hadoop expertise that operates a large Hadoop cluster for providing personalized recommendations of content to their users. However, the batch-oriented nature of Hadoop has so far prevented them from using that Hadoop infrastructure for real-time ad serving, streaming analytics on advertisement logs, and providing real-time feedback on advertisement campaign performance to their customers. Also, for their business and marketing analysts, who want to perform ad hoc queries on the advertising data to target a larger pool of users, they have set up a separate data repository away from both the real-time analytics and batch analytics systems. As a result, the operating expenses of their data infrastructure have more than tripled. Worse, they incur a huge overhead, just trying to keep the data synchronized across these three platforms. Since essentially the same data is kept in multiple places, there is a lag and discrepancies in the data and repetitive tasks for data cleansing, ensuring data quality, and maintaining data governance waste more than 80% of precious and valuable time of their data scientists and big data infrastructure specialists.

Acme architects and engineers decided to replace the entire data infrastructure with modern flash- and memory-based architecture. However, because of the high costs of these modern hardware systems, they are unsure whether the return on investment will justify these increased capital expenditures. In addition, they need to consider the engineering costs of rewriting their entire existing data pipelines, written over more than 5 years, for these modern architectures. There are also considerations of having to retrain the data practitioners for utilizing new technologies. Based on process considerations, they decided to do incremental, piecemeal upgrades to their data infrastructure, moving toward a more unified data processing platform from their current disparate systems. However, evaluating modern infrastructures, without having to rewrite the entire data pipeline, would require a standard benchmark that is necessary for Acme to evaluate new systems.

Datasets

There are four important large datasets used in Acme's data analysis pipeline.

User Profiles

User profiles contain details about every registered user. The schema for user profile is as follows:

- UserID: UUID (64–128 bit unique identifier)
- Age: 0.255
- Sex: M/F/Unknown
- Location: Lat-Long
- Registration time stamp: TS
- Interests: Comma separated list of (topic: subtopic:keyword)

Advertisements

This dataset contains all the details about all the advertisements available for displaying within content. The schema for this dataset is as follows:

- AdID: UUID
- CampaignID: UUID
- CustomerID: UUID
- AdType: “banner,” “modal,” “search,” “video,” “none”
- AdPlatform: “web,” “mobile,” “both”
- Keywords: comma separated list of (topic: subtopic:keyword)
- PPC: \$ per click
- PPM: \$ per 1000 ads displayed
- PPB: \$ per conversion

Content Metadata

Content dataset contains all the metadata about the content. The schema for the content dataset is as follows:

- ContentID: UUID
- ContentType: “news,” “video,” “audio,” “photo”
- Keywords: comma separated list of (topic: subtopic:keyword)

Ad Serving Logs

This dataset is streamed continuously from the ad servers. Each entry in this log has the following fields, of which some may be null:

- TimeStamp: TS
- IPAddress: IPv4/IPv6
- UserID: UUID
- AdID: UUID
- ContentID: UUID
- AdType: “banner,” “modal,” “search,” “video”
- AdPlatform: “web,” “mobile”
- EventType: “View,” “Click,” “Conversion”

Computations

The following computational steps are performed on the data in Acme’s advertisement analytics data pipelines.

Ingestion and Streaming Analytics

This phase of the data pipeline is based on a popular streaming analytics benchmark proposed by a large Ad-Tech company. Ad servers produce ad click, view, and conversion events to a message queue, or a staging storage system. Queue consumers consume events continuously from a message queue and process them in a streaming manner using a streaming analytics platform. For every event consumed, the following computations are performed:

1. Parse the event record
2. Extract time stamp, AdID, EventType, and AdType
3. Look up CampaignID from AdID
4. Windowed aggregation of event types for each AdID and CampaignID
5. Store these aggregates in an aggregate dataset
6. Prepare these aggregations for a streaming visualization dashboard for a CampaignID and all Ads in that Campaign

The two output datasets from the streaming analytics stage are:

1. AdID, Window, nViews, nClicks, nCon, \sum PPV, \sum PPC, \sum PPCon
2. CmpgnID, Window, nViews, nClicks, nCon, \sum PPV, \sum PPC, \sum PPCon

Second streaming ingestion pipeline keeps the User Table, Ad Table, and Content Table updated. While the ads are being displayed, clicked, and converted, new users are being registered, and existing users’ information is being updated. New campaigns are created, existing campaigns are modified, new ads are being created, and existing ads are updated. The correct implementation of this data pipeline will allow these inserts and updates taking place concurrently with other stages of the pipeline, rather than periodically, thus introducing transactionality. In this ingestion pipeline also, we use message queue consumers to get insert and update records and apply these inserts and updates to respective datasets in storage in real time. The steps in this pipeline are as follows:

1. Ingest a {user, campaign, ad} {update, insert} event from message queue.
2. Parse the event to determine which dataset is to be updated.
3. Update respective dataset.
4. Keep track of total number of updates for each dataset.
5. When 1% of the records are either new or updated, launch the batch computation stage described below and reset update counters.

Batch Model Building

In this batch-oriented computational stage, we build ad targeting models. The inputs for this pipeline are the user dataset, ad dataset, and content dataset. And outputs of this pipeline are two new datasets:

1. ($UserID$, $AdID_1$, $weight_1$, $AdID_2$, $weight_2$, $AdID_3$, $weight_3$)
2. ($ContentID$, $AdID_1$, $weight_1$, $AdID_2$, $weight_2$, $AdID_3$, $weight_3$)

These datasets represent the top 3 most relevant ads for every user and for every content. These datasets are then used for the ad serving systems, such that when a user visits a particular content, the best match among these ads is chosen, based on one look up each in the user dataset and content dataset.

Relevance of an ad for a user or a content is determined by cosine similarity in the list of keywords, topics, and subtopics. This model building pipeline, built using batch-oriented computational frameworks, has the following steps:

1. Extract the relevant fields from user dataset and ad dataset, and join based on topics, subtopics, and keywords.
2. Filter the top 3 matching keywords, and compute the weights of ads for those keywords using cosine similarity.
3. Repeat the steps above for the content dataset and ad dataset.

Interactive and Ad Hoc SQL Queries

The interactive and ad hoc queries are performed on varying windows of the aggregates for campaigns and ads using an interactive query engine (preferably SQL-based). Some examples of the queries are:

1. What was the {per minute, hourly, daily} conversion rate for an ad? For a campaign?
2. How many ads were clicked on as a percentage of viewed, per hour for a campaign?
3. How much money does a campaign owe to Acme for the whole day?
4. What are the most clicked ads and campaigns per hour?
5. How many male users does Acme have aged 0–21 or 21–40?

The results of these queries can be displayed in a terminal, and for the queries resulting in time-windowed data, visualized using BI tools.

Various stages of computations in the Ad-Bench data pipeline and dataflow among them are shown in Fig. 1.

Scale Factors and Metrics

When compared to microbenchmarks or benchmarks that consist of a fixed set of queries representing similar workloads, performed on a single system, an end-to-end data pipeline with mixed workloads poses significant challenges in defining scale factors and metrics and reporting benchmark results. In this section, we propose a few alternatives, with reasoning behind our choice.

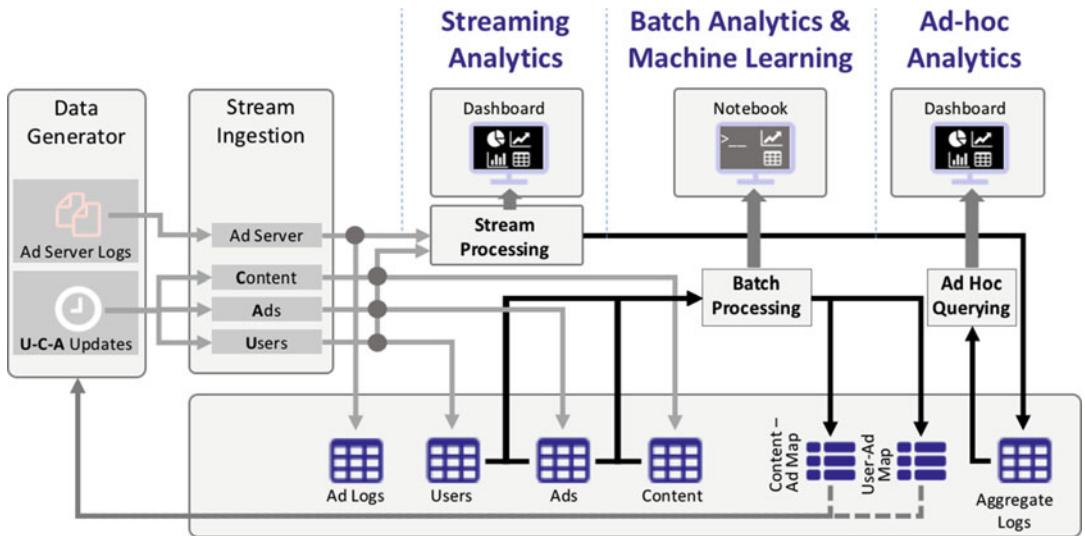
Scale Factors

Across different verticals, the number of entities and thus the amount of data and data rates vary widely. For example, in financial institutions, such as Banks providing online banking, the number of customers (users) is between a few tens of thousands and millions, the number of advertisements that correspond to the number of different services (such as loans, fixed deposits, various types of accounts) is tens to hundreds, the number of content entities is equivalent to the number of dashboards (such as account transaction details, bill pay), and the data rates tend to be in low tens of thousands per minute even during peak times. These scales are very different than a content aggregation website or mobile application. To address these varying needs for various industries, we suggest the following scale factors:

Class	Users	Ads	Contents	Events/second
Tiny	100,000	10	10	1,000
Small	1,000,000	100	100	10,000
Medium	10,000,000	1,000	1,000	100,000
Large	100,000,000	10,000	10,100	1,000,000
Huge	1,000,000,000	100,000	100,000	10,000,000

Metrics

Since the benchmark covers an end-to-end data pipeline consisting of multiple stages, measures of performance in each of the stages vary widely. Streaming data ingestion is measured in terms of number of events consumed per second, but due to varying event sizes, amount of data ingested



End-to-End Benchmark, Fig. 1 Dataflow across AdBench data pipeline

per second is also an important measure. Streaming data analytics operates on windows of event streams, and performance of streaming analytics should be measured in terms of window size (amount of time per window, which is a product of number of events per unit time, and data ingestion rate). Because of the latency involved in ingestion of data and output of streaming analytics, the most relevant measure of performance of streaming analytics is the amount of time between event generation and event participation in producing results.

Batch computations, such as machine-learned model training (of which recommendation engine is a representative example), the amount of time needed from the beginning of the model training, including any preprocessing of raw data, to the upload of these models to the serving system, need to be considered as performance measure.

Similarly, for ad hoc and interactive queries, time from query submission to the production of results is the right performance measure.

While most of the benchmark suits attempt to combine the performance results into a single performance metric, we think that for an end-to-end data pipeline, one should report individual performance metrics of each stage, to enable choice of different components of the total technology stack. For a rapidly evolving

big data technology landscape, where a complete data platform is constructed from a collection of interoperable components, we encourage experimentation of combining multiple technologies to implement end-to-end data pipeline, and stage-wise performance metrics will aid this decision.

In order to simplify comparisons of environments used to execute these end-to-end pipelines, initially, a simple weighted aggregation of individual stage's performance is proposed as a single performance metric for the entire system, with weights explicitly specified. There is anecdotal evidence to suggest that the business value of data decreases as time between production and consumption of data increases. Thus, one might be tempted to give higher weight to the streaming computation performance than to the batch computation performance. However, it also might be argued that approximate computations in real time are acceptable, while batch computations need exact results, and therefore faster batch computations should be given higher weightage. We propose that determining appropriate weights for individual stages should be an active area of investigation and should be incorporated in the metrics based on the outcome of this investigation.

In addition to performance metrics, the complexity of assembling data pipelines by

combining multiple technologies must be taken into account by the architects of these pipelines. Skills needed to effectively use multiple technologies and operational complexity of these data pipelines tend to be some of the important factors in choosing such systems. These are reflected in the total cost of ownership (TCO) of the system under test. We recommend that metrics such as time and manpower required to develop the pipeline, time to deployment, management, and monitoring tools for operationalizing data pipelines should be reported.

References

- Baru C, Bhandarkar M, Nambiar R, Poess M, Rabl T (2013) Benchmarking big data systems and the big data top100 list. *Big Data* 1(1):60–64
- Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen HA (2013) Bigbench: towards an industry standard benchmark for big data analytics. In Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD’13, ACM, New York, pp 1197–1208

Energy Benchmarking

Klaus-Dieter Lange¹ and
Jóakim von Kistowski²

¹Hewlett Packard Enterprise, Houston, TX, USA

²Institute of Computer Science, University of Würzburg, Würzburg, Germany

Synonyms

Energy-efficiency benchmarking; Power benchmarking

Definitions

Software that utilizes standard applications, or their core routines that focus on a particular access pattern, in order to measure the power

consumption or energy efficiency of single or multiple servers or storage systems.

Overview

Energy benchmarks are useful tools to analyze the power consumption and efficiency of IT equipment. This chapter will provide historical context for energy benchmarking, brief descriptions of the most commonly used benchmarks, current research findings, and future research challenges.

Historical Background

In 1996, one of the earliest works on an energy-efficiency metric was published by Gonzalez and Horowitz (1996) in which the authors proposed the energy-delay product as the metric of energy-efficient microprocessor design. Almost a decade later, January 2006, the SPECpower Committee (<https://www.spec.org/power>) was founded and started the development of the first industry-standard energy-efficiency benchmark for server-class compute equipment.

The SPECpower Committee developed the SPEC PTDAEMON (http://www.spec.org/power/docs/SPEC-PTDaemon_Design.pdf) interface which offloads the details of controlling a power analyzer or temperature sensor, presenting a common TCP/IP-based interface that can be readily integrated into different benchmark harnesses.

The Power and Performance Benchmark Methodology (https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf) was created by the SPECpower Committee for performance benchmark designers and implementers who want to integrate a power component into their benchmarks. This public document also serves as an introduction to those who need to understand the relationship between power and performance metrics in computer systems benchmarks. Guidance is provided for including power metrics in existing benchmarks, as well as altering existing benchmarks and

designing new ones to provide a more complete view of energy consumption.

In 2007, the research benchmark JouleSort (Rivoire et al. 2007) was released and later that year, the industry standard SPECpower_ssj2008 benchmark (http://spec.org/power_ssj2008), which exercises the CPUs, caches, memory hierarchy, and the scalability of shared memory processors on multiple load levels (Lange 2009). The benchmark runs on a wide variety of operating systems and hardware architectures. 2008 marks the first release of the SPC Energy specifications (<http://spcresults.org/specifications>) and the TPC-Energy specification ([http://www\(tpc.org/tpc_energy/default.asp](http://www(tpc.org/tpc_energy/default.asp)), which utilizes the SPEC PTDaemon.

The SPECpower Committee augmented the SPECpower_ssj2008 benchmark with multi-compute node support in 2009 and started the work on the SERT (<https://www.spec.org/sert>) suite by designing the Chauffeur (<https://www.spec.org/chauffeur-wdk>) framework, followed with the implementation of the workload, automated HW and SW discovery, and GUI.

Several enhancements and code changes to all SPECpower_ssj2008 benchmark components were included in subsequent releases in 2011 and 2012. SPEComp2012 (<https://www.spec.org/omp2012>) and the vendor-based benchmarks VMmark (<https://www.vmware.com/products/vmmark.html>) and SAP (<https://www.sap.com/about/benchmark.html>) were also released in 2012, utilizing the SPEC PTdaemon and the SPEC benchmark methodology.

After a series of successful alpha and beta test phases, the SERT suite was released in 2013 and 2 months later, the US Environmental Protection Agency (EPA) adopted the SERT suite for their ENERGY STAR for Server Certification program. Over the years, four SERT upgrades were released in order to further enhance the usability and to increase its platform support. SPECvirt_sc2013 (https://www.spec.org/virt_sc2013) was also released that year replacing its predecessor, SPECvirt_sc2010.

The Chauffeur framework from SERT is made available as the Chauffeur Worklet Development Kit (WDK). This kit can be used to develop new

workloads (or “worklets,” in Chauffeur terminology). Researchers can also use the WDK to configure worklets to run in different ways, in order to mimic the behavior of different types of applications. These features can be used in the development and assessment of new technologies such as power management capabilities. The first version of the Chauffeur WDK was released in 2014, and succeeded by releases in 2015 and 2017.

For the SERT suite 2.0, one of the main focuses was the reduction of its run time and improvements on the automated hardware and software discovery, in order to save the user testing time and costs. The support of additional power analyzers and interfaces was adopted as well in order to provide wider choices for users. Also, some major improvements of the memory subsystem worklets for more accurate characterization of memory were implemented. Simultaneously to these efforts, the SPECpower Committee, in collaboration with the more academic-focused SPEC Power Research Group (<https://research.spec.org/working-groups/rg-power.html>), conducted large-scale experiments in order to determine the server efficiency metric (<https://www.spec.org/sert2/SERT-metric.pdf>) for SERT Suite 2.0, which enabled its release on March 28, 2017.

Foundations

Energy benchmarks are useful tools to analyze the power consumption and efficiency of IT equipment. They execute standard applications or specialized micro-workloads to stress a system, enabling measurement of the system’s power consumption. Workloads used within an energy benchmark can be adopted from existing benchmarks. Specific to the energy benchmarks are measurement methodologies regarding power measurement and metrics that combine the energy measurement result with the primary performance metric of the underlying benchmark.

An energy benchmark measurement methodology defines, among other things, the point

at which energy is measured. This point is usually the entire system for application benchmarks, but may be specifically instrumented system components for micro- or component-benchmarks. System component instrumentation is rare. Most benchmarks instrument the entire system under test, as it enables comparison of different system architectures and require the least constraints on internal system structure for the benchmark. To further improve the comparability of measurement results, power benchmark measurement methodologies may impose restrictions and requirements on the accuracy of power measurement devices to be used in the benchmark (e.g., the SPEC Power and Performance Methodology (https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf) requires a maximum measurement uncertainty of 1% for all used power measurement devices).

In addition to defining how power is measured, an energy benchmark often defines a performance measure and measurement method. This performance measure usually is intended to be used in conjunction with the power measurements in order to derive an efficiency metric. For this context, *throughput* has established itself as the go-to metric. Throughput defines itself as the number of completed work units per time:

$$\text{Throughput} = \frac{\text{Completed Work Units}}{\text{Measurement Time}} \left[\frac{1}{\text{s}} \right]$$

As work units vary between different workloads, throughput is often normalized by comparing it to the throughput of a reference system.

Energy efficiency is computed using an energy-efficiency metric, which constructs a ratio of power consumption and performance, such as

$$\text{Efficiency} = \frac{\text{Performance}}{\text{Power Consumption}}$$

Alternatively, efficiency may be a ratio of work performed, which is often (depending on the concrete metric) mathematically equivalent to performance per power consumption:

$$\text{Efficiency} = \frac{\text{Work performed}}{\text{Energy expended}}$$

Using throughput as the performance metric, energy efficiency is the ratio of throughput and power consumption in Watts. This is mathematically equal to the alternative efficiency ratio, as work units per time divided by power equals work units divided by energy:

$$\begin{aligned} \text{Efficiency} &= \frac{\text{Throughput}}{\text{Power Consumption}} \left[\frac{\text{s}^{-1}}{\text{W}} \right] \\ &= \frac{\text{Work Units}}{\text{Energy expended}} \left[\frac{1}{\text{J}} \right] \end{aligned}$$

Energy benchmarks can be used to provision Big Data systems with hardware or/and underlying software stack that reduces operational costs and meets environmental criteria.

The following are descriptions of the most common energy benchmarks and energy benchmark specifications:

JouleSort is probably the oldest energy-efficiency benchmark, which is also intended for use with large-scale server systems, as employed in Big Data environments. It was released in 2007 and rates energy efficiency by measuring the total energy consumed while sorting a large array of fixed sizes. It features three workload classes, which differ in the size of the array to be sorted (10 GB, 100 GB, and 1 TB).

SPECpower_ssj2008 is the first industry-standard benchmark for server energy efficiency. It measures the energy efficiency of an emulated transactional enterprise application. SSJ's innovation is the measurement of energy efficiency at multiple load levels. It computes energy efficiency by dividing its normalized throughput (transactions per second) by power consumption. The final score is the sum of the separate load level scores. SSJ is relevant for Big Data, as its methodology has been adopted by many of the following benchmarks.

SPC Energy Specifications are specifications for the benchmarks of the Storage Performance Council (SPC). These benchmarks test the

performance of storage systems (SPC-1 and SPC-2) and storage systems components (SPC-1C and SPC-2C). The energy specifications specify modes of benchmark execution for which power consumption is measured. Power consumption is measured at three loads (idle, mid, and high) and reported with the performance results of the regular benchmark run.

TPC Energy Specification features explicit rules for energy measurement as part of the TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks. TPC energy requires measurement of power consumption of the entire system under test and then computes an efficiency metric corresponding to each benchmark's performance metric. Analogue to the TPC's pricing metrics, the efficiency metric divides power consumption by performance, i.e., computing the inverse metric to the regular efficiency metric used by other benchmarks.

SERT is the Server Efficiency Rating Tool developed by SPEC to evaluate the energy efficiency of servers. The SERT suite is a suite of 12 “worklet” modules that stress different system components and measure performance as well as the power consumption of the system. These worklets stress the CPU, memory, and disk I/O subsystem at various target load levels just as well as one hybrid worklet that stresses a series of complex CPU and memory access patterns. The SERT suite utilizes the Chauffeur harness and is configurable so individual worklets or categories of worklets can be run.

SPECvirt_sc2013 measures the server consolidation aspect of virtualized environments, replacing the SPECvirt_sc2010 benchmark. Its workload is comprised of components from the SPECweb2005, SPECjAppserver2004, SPECmail2008, and SPEC CPU2006 benchmarks. The benchmark reports the performance (SPECvirt_sc2013); performance/power for the system under test including storage (SPECvirt_sc2013_PPW); and performance/power for the server-only (SPECvirt_sc 2013_ServerPPW).

SPEComp2012 is a benchmark package developed by the Standard Performance Evaluation

Corporation (SPEC) to provide a comparative measure of compute-intensive performance across the wide practical range of HPC platforms. The benchmark uses 14 workloads developed from real-world programs from a variety of application areas, including molecular dynamics, fluid dynamics, physics, weather-forecasting, imaging, and computer development. The SPEComp2012 benchmark reports performance in SPECcompG_base2012/SPECcompG_peak2012 (the normalized geometric mean of all 14 workloads) and energy in SPECcompG_energy_base2012/SPECcompG_energy_peak 2012.

VMmark is a benchmark from VMware used to measure the performance of multi-node VMware environments. It introduced power measurement with version 2.5 and utilized the SPEC PTDaemon as the interface between the power analyzer and the benchmark harness. VMmark's workload leverages components from several application areas (Exchange 2007, Olio, DVD Store 2, and a standby mode) in order to stress the system with common virtualization procedures, e.g., virtual machine relocation, relocation, cloning, deployment, and load balancing. VMmark reports the power consumption in power-per-kilowatt (PPKW).

SAP is a benchmark from SAP SE used to measure the performance of SAP application environments. In 2009, the SAP server power benchmark was added to the SAP Standard Application Benchmark Suite. It reports performance in SAP Application Performance Standard (SAPS) and energy efficiency as a key performance indicator (kpi), representing the amount of power consumption per 1000 SAPS (W/kSAPS).

Key Research Findings

A significantly large class of research in Big Data, such as the research in Mashayekhy et al. (2015), focuses on placement of jobs (usually MapReduce jobs) on servers. This kind of

research can utilize energy benchmark results for the information that it provides on the systems in question.

Other research focuses on energy management of the servers themselves when running Big Data applications (Chen et al. 2014). Energy benchmarks are used for evaluating the management mechanisms that result from such research.

Research on the benchmarks themselves has been conducted with focus on workloads and load levels. Load levels have been introduced with the SPECpower_ssj2008 benchmark and have since been implemented in the SPEC SERT suite as well. As load levels are intended to increase the relevance of the energy benchmarks using them, research on these levels has focused on the relevance aspect. In addition, research has investigated the reproducibility of load level implementations. An analysis in Kistowski et al. (2016) showed that the load level mechanism of SPECpower_ssj2008 and SERT allows for a high reproducibility regarding measurement re-runs. Workloads, on the other hand, are usually application specific, but the investigation in Kistowski et al. (2015) demonstrated explicitly that benchmark suites that employ multiple workloads can provide more information for server characterization. This analysis also investigated the information gain that can be achieved by increasing the number of load levels. Similarly, results from multiple micro-benchmarks have been used by researchers for the creation of system models (Bellosa 2000), which, in turn, can be used for energy management research.

Future Directions for Research

Reproducibility is a significant challenge for energy benchmarks, as these benchmarks require both the performance results, as well as the power measurements to be reproducible. However, it was shown that power measurements of nominally identically servers, using hardware with the same nameplate information and configuration, can differ for as much as 15% (Chen et al. 2014). This difference is due primarily to differences in semiconductor quality within the hardware itself.

Future research on energy-efficiency benchmarks must find answers on how to deal with this spread of results.

Cross-References

- ▶ [Energy Efficiency in Big Data Analysis](#)
- ▶ [Energy Implications of Big Data](#)

References

- Bellosa F (2000) The benefits of event-driven energy accounting in power-sensitive systems. In: Proceedings of the 9th workshop on ACM SIGOPS European workshop, pp 37–42 See ACM link: <https://dl.acm.org/citation.cfm?id=566736>
- Chauffeur framework. <https://www.spec.org/chauffeur-wdk>
- Chen M, Mao S, Liu Y (2014) Big data: a survey. *Mob Netw Appl* 19:171. <https://doi.org/10.1007/s11036-013-0489-0>
- Gonzalez R, Horowitz M (1996) Energy dissipation in general purpose microprocessors. *IEEE J Solid State Circuits* 31(9):1277–1284
- Lange K-D (2009) Identifying shades of green: the SPECpower benchmarks. *Computer* 42(3):95–97. IEEE Computer Society Press, Los Alamitos
- Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W (2015) Energy-aware scheduling of MapReduce jobs for big data applications. *IEEE Trans Parallel Distrib Syst* 26(10):2720–2733
- Rivoire S, Shah MA, Ranganathan P, Kozyrakis C (2007) JouleSort: a balanced energy-efficiency benchmark. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data. SIGMOD '07. ACM, New York, pp 365–376
- SAP. <https://www.sap.com/about/benchmark.html>
- Server Efficiency Rating Tool (SERT). <https://www.spec.org/sert>
- SPC specifications. <http://spcresults.org/specifications>
- SPEC power and performance methodology. https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf
- SPEC PTDaemon interface. http://www.spec.org/power/docs/SPEC-PTDaemon_Design.pdf
- SPEC RG Power Working Group. <https://research.spec.org/working-groups/rq-power.html>
- SPEComp2012. <https://www.spec.org/omp2012>
- SPECpower Committee. <https://www.spec.org/power>
- SPECpower_ssj2008. http://spec.org/power_ssj2008
- SPECvirt_sc2013. https://www.spec.org/virt_sc2013
- The SERT 2 metric. <https://www.spec.org/sert2/SERT-metric.pdf>
- TPC Energy. http://www.tpc.org/tpc_energy/default.asp

- VMmark. <https://www.vmware.com/products/vmmark.html>
- von Kistowski J, Block H, Beckett J, Lange K-D, Arnold JA, Kounev S (2015) Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads. In: Proceedings of the 6th ACM/SPEC international conference on performance engineering (ICPE '15). ACM, New York, pp 223–234
- von Kistowski J, Block H, Beckett J, Spradling C, Lange K-D, Kounev S (2016) Variations in CPU power consumption. In: Proceedings of the 7th ACM/SPEC on international conference on performance engineering (ICPE '16). ACM, New York, pp 147–158

Energy Efficiency in Big Data Analysis

Carmela Comito
CNR-ICAR, Rende, Italy

Overview

Data generation has increased drastically over the past few years. Processing large amounts of data requires huge compute and storage infrastructures, which consume substantial amounts of energy. Moreover, another important aspect to consider is that more and more the data is analyzed on-board battery operated mobile devices like smart-phones and sensors. Therefore, data processing techniques are required to operate while meeting resource constraints such as memory and power to prolong a mobile device network's lifetime. This chapter reviews representative methods used for energy efficient Big Data analysis, providing first a generic overview of the issue of energy conservation and then presenting a more detailed analysis of the issue of energy efficiency in mobile and sensor networks.

Energy-Aware Big Data Analysis: Approaches and Trends

Distributed data analysis strategies are widely adopted to cope with huge volumes of data and to provide fast response times. Usually distributed

data analysis is performed on computer clusters composed by thousands of computers and hosted in large data centers. While such facilities enable large-scale online services, they also raise economical and environmental concerns. Indeed, a large-scale data center can draw tens of megawatts of electricity to operate, and it can cost millions of US dollars per year in terms of energy expenditure. Therefore, an important problem to address is how to reduce the energy expenditure of data centers. Obviously, a possible solution to these challenges consists in designing more energy-efficient data centers, which consume less energy and, consequently, pollute and cost less. In the past, a large part of the energy consumption of a data center could be accounted to inefficiencies in its cooling and power supply systems. However, many companies already adopt state-of-the-art techniques to reduce the energy wastage of such systems, leaving little room for more improvements in those areas. Indeed, the energy consumption of a state-of-the-art data center would be reduced by less than 24% if all the overheads in its cooling and power supply systems were eliminated. Therefore, new approaches are necessary to mitigate the environmental impact and the energy expenditure of data centers.

This section reports the main approaches proposed to reduce energy consumption of big data analysis frameworks, by differentiating among the hardware and software solutions. At the end of the section, the few initiatives that have been done in the DBMS community are outlined.

Hardware Approaches

The energy expenditure of data centers can be mitigated by reducing the energy consumption of the computing resources. In fact, most of the existing research work in the area of energy-aware systems concerns hardware-based techniques focused on reducing the energy consumption of the processor. Among those techniques, turning off idle components is widely adopted (Li et al. 1994). In particular, CPUs are the most energy-consuming component in servers dedicated to query processing, accounting for 40% of total energy consumption when a server is idle and for

66% of total energy consumption when it is fully utilized.

Dynamic Voltage and Frequency Scaling (DVFS) is another hardware-based technique for energy conservation, allowing for simultaneously varying the processor voltage and frequency as per the energy performance level required by the tasks (Zhuo and Chakrabarti 2005; Aydin et al. 2004; Seth et al. 2003). In fact, higher core frequencies mean faster computations but higher power consumption, while lower frequencies lead to slower computations but reduced power consumption. However, arbitrary slowing down the CPU can yield significant performance degradation especially during compute-bound application phases: low frequencies entail longer query processing times that may be unacceptable for the users.

Another important aspect that can be exploited to reduce the energy consumption of a server is the fact that users can hardly notice response times that are faster than their expectations. Therefore, sometimes it is not necessary to process queries faster than user expectations, and based on this observation, Catena and Tonellootto (2017) proposed the Predictive Energy Saving Online Scheduling Algorithm (PESOS) to select the most appropriate CPU frequency to process a query on a per-core basis. PESOS aims at process queries by their deadlines and leverage high-level scheduling information to reduce the CPU energy consumption of a query processing node. PESOS bases its decision on query efficiency predictors, estimating the processing volume and processing time of a query.

Uni-processor energy-aware scheduling algorithms have been developed to make an efficient use of hardware-based energy management techniques. An energy-aware scheduling algorithm, proposed in Weiser et al. (1996), orders task execution such that different components of a mobile computing device can have longer idle periods to be shut down. Several energy-saving scheduling (e.g., Zhang et al. 2002; Luo and Jha 2000) algorithms for multi-processor distributed systems are focused on architectures with dynamic voltage scaling or dynamic power management capabilities.

Software Approaches

Among software-based energy-aware techniques, remote execution provides that a device with limited energy transfers a computational task to a nearby device which is more energy powerful (Rudenko et al. 1998; Li et al. 2001). Software-based energy management emerged as a key topic for resource management in distributed cluster-based systems (Bianchini and Rajaniony 2004; Lefurgy et al. 2003). The use of virtualization for consolidation is presented in Petrucci et al. (2009), which proposes an approach for power optimization in virtualized server clusters through an algorithm that dynamically manages the virtualized servers. Following the same idea, Liu et al. (2009) aims to reduce the power consumption in virtualized data center by supporting task migration and task placement optimization. Verma et al. (2008) also propose a virtualization-aware adaptive consolidation approach, measuring energy costs executing a given set of applications and using correlation techniques to predict usage. PARM (Mohapatra and Venkatasubramanian 2003) is a distributed power-aware middleware framework consisting of several distributed servers (service providers), proxy servers, metadata repositories (directory services), and mobile clients. The framework adapts to the diminishing power availability of the devices over time, by dynamically off-loading expensive middleware components to a proxy.

Energy-aware task scheduling is another software method where the scheduling policy aims at optimizing the energy consumption. Alsalih et al. (2005) proposed an energy-aware dynamic task allocation algorithm over mobile networks. They adopt an objective function that minimizes the energy consumption in the network. The solution proposed in Alsalih et al. (2005) does not address the communication aspects of the system; it considers only the local computation issues and works at a node level without taking into account the workload in the rest of the network.

DBMS Approaches

With the rising energy costs and energy-inefficient server deployments, it is key for

DBMSs to evolve and to consider energy efficiency as a main goal (Lang et al. 2011).

Processing the query as fast as possible is not the most energy-efficient way to operate. First, typical servers often run at low utilization. This opens the opportunity to execute the query slower, if the additional delay is acceptable. For example, this situation may occur if the service-level agreement (SLA) permits the additional response time penalty. Since typical SLAs are written to meet peak or near-peak demand, SLAs often have some slack in the frequent low utilization periods, which could be exploited by the DBMS. Second, components on modern server motherboards offer various power/performance states that can be manipulated from software. Of these components, the CPU is currently the most amenable for transitioning between such power/performance states. Based on these observations, in Lang et al. (2011) is proposed a novel framework where the query optimization problem is to find and execute the most energy-efficient plan that meets the SLA. To enable this framework, existing query optimizers have been extended with an energy consumption model, in addition to the traditional response time model. With these two models, the enhanced query optimizer can generate a structure that details the energy and response time cost of executing each query plan at every possible power/performance state under consideration.

Lang and Patel (2009) proposed the ecoDB project whose goal is to investigate energy-efficient methods for data processing in distributed computing environments, proposing two mechanisms that can trade energy for performance. The first mechanism, called Processor Voltage/Frequency Control (PVC), lets the DBMS explicitly change the processor voltage and frequency parameters. Processors can take commands from software to adjust their voltage/frequencies to operate at points other than their peak performance. The second mechanism is at the workload management level. It is referred to as the Explicit Query Delays (QED) approach where queries are delayed and placed into a queue on arrival. When the queue reaches a certain threshold, all the queries in the

queue are examined to determine if they can be aggregated into a small number of groups, such that the queries in each group can be evaluated together. Evaluating this workload can trade average query response time for reduced overall energy consumption. Single group queries can be run in the DBMS at a lower-energy cost than the individual queries.

In Roukh et al. (2016) is proposed an energy-aware query optimizer framework, called EnerQuery, which leverages the PostgreSQL query optimizer by integrating the energy dimension. EnerQuery is built on top of a traditional DBMS to capitalize the efforts invested in building energy-aware query optimizers. Energy consumption is estimated on all query plan steps and integrated into a mathematical linear cost model used to select the best query plans.

E

Energy-Efficient Data Analysis on Mobile Computing

The increasing power of wireless devices is opening the way to support analysis and mining of data in a mobile context (Wang et al. 2003).

Mobile data analysis and mining may include different scenarios in which a mobile device can play the role of data producer, data analyzer, client of remote data miners, or a combination of them. Accordingly, an increasing number of smartphone and PDA-based data intensive applications have been recently developed (Kargupta et al. 2002, 2003; Wang et al. 2003). Examples include smartphone-based systems for body-health monitoring, vehicle monitoring, and wireless security systems.

A key aspect that must be addressed to enable effective and reliable data mining over mobile devices is ensuring energy efficiency, as most commercially available mobile devices have battery power which would last only a few hours (Comito et al. 2011, 2012).

In Bhargava et al. (2003), based on the experimental energy consumption data collected for several popular data analysis techniques, it has introduced energy efficiency as a novel criterion

to evaluate algorithms intended to run on mobile and embedded platforms. The authors characterize and compare the energy consumed by data mining applications implemented onboard a device with the energy costs for sending the data to a remote site. The obtained results show that for low-bandwidth lossy networks, the high energy costs of communication often make local onboard data mining a more energy-efficient choice.

The work in Comito and Talia (2017) enables energy-efficient execution of data mining algorithms on mobile devices, making them lightweight in terms of resource utilization. To achieve this purpose, the approach is to act on data dimensionality according to the fact that mining algorithms are data intensive. Therefore, in order to identify the key factors in the data complexity in Comito and Talia (2017), an experimental study of the energy consumption characteristics of representative data mining algorithms running on mobile devices is performed. The performance analysis depends on many factors encompassing size of data sets, attribute selection, number of instances, number of produced results, and accuracy. To carry out the performance evaluation, an Android application has been developed allowing users to execute data mining algorithms over Android smartphones by tuning a set of parameters. The energy characterization outcomes have been then exploited to predict the energy costs of data mining algorithms running on mobile devices. Specifically, in Comito and Talia (2017) is designed and implemented a machine learning-based framework to predict, from a set of mobile devices and set of data mining tasks, the resulting energy consumption of such tasks over the mobile devices. A supervised approach has been used where the main idea is to exploit examples of cases from past system behavior to construct an energy model that is able to predict the class of new examples.

Many emerging context-aware mobile applications involve the execution of continuous queries over sensor data streams generated by a variety of onboard sensors on multiple personal mobile devices. To reduce the energy overheads,

the work in Yang et al. (2013) introduces CQP, a collaborative query processing framework that exploits the overlap across multiple smartphones. The framework automatically identifies the shareable parts of multiple executing queries and then reduces the overheads of repetitive execution and data transmissions, by having a set of “leader” mobile nodes executing and disseminating these shareable partial results. To further reduce energy, CQP utilizes lower-energy short-range wireless links to disseminate such results directly among proximate smartphones.

Efficient resource allocation and energy management can be achieved through clustering of mobile nodes into local groups. In Comito and Talia (2014) is proposed a clustering scheme where mobile devices are organized into local groups (clusters or mobile groups). Each cluster has a node referred to as cluster head that acts as the local coordinator of the cluster. The focus of the approach is building and maintaining a cluster structure in a network of cooperating mobile devices. The problem of assigning devices to clusters has been formulated as a multicriteria decision-making problem based on the weighted-sum approach. A multi-objective cost function is defined where the aim of the optimization is to maximally extend the network lifetime and the mobility similarity of the devices within a group. Depending on the specific application scenario, the weights associated to the two costs can be properly tuned. This way optimal decisions may be taken in the presence of trade-offs between the two objectives. A weighted metric that combines the effect of energy and mobility of nodes is also introduced to select suitable cluster-head nodes.

In Comito et al. (2017) is proposed an energy-aware (EA) scheduling strategy for allocating computational tasks over a wireless network of mobile devices. In such a network, the devices are clustered into local groups, named *clusters*. Generally, geographically adjacent devices are assigned to the same cluster. To make the most of all available resources, the EA strategy tries to find a suitable distribution of tasks among clusters and individual devices. Specifically, the

main design principle of the strategy is finding a task allocation that prolongs the total lifetime of the wireless network and maximizes the number of alive devices by balancing the energy load among them. To this end, the EA scheduler implements a two-phase heuristic-based algorithm. The algorithm first tries to assign a task locally to the cluster where the execution request has been generated, so as to maximize the cluster residual life. If the task cannot be assigned locally, the second phase of the algorithm is performed by assigning the task to the most suitable node all over the network of clusters, maximizing this way the overall network lifetime. The energy model takes into account the energy costs of both computation and communication.

Energy-Efficient Query Processing in Sensor Networks

Query processing has been studied extensively in traditional database systems. However, few existing methods can be directly applied to wireless sensor database systems (WSDSs) due to their characteristics, such as decentralized nature, limited computational power, imperfect information recorded, and energy scarcity in individual sensor nodes. Moreover, traditional database systems result in high energy consumption and low energy efficiency due to the lack of consideration of energy issues and environmental adaptation in the design process. In this section, representative recent efforts on this issue, with a focus on energy-aware query optimization and energy-efficient query processing, are reported.

In Guo et al. (2017) is proposed a method of modeling energy cost of query plans during query processing based on their resource consumption patterns, which helps predict energy cost of queries before execution. Using the cost model as a basis, the evaluation model can utilize the trade-offs between power and performance of plans and helps the query optimizer select plans that meet performance requirements but result in lower-energy cost. Finally, a green database framework integrated with the two above models is proposed to enhance a commercial DBMS.

Experimental results reveal that, with reliable and accurate statistical data, the framework proposed in this study can achieve significant energy savings and improve energy efficiency.

Query for sensor networks should be wisely designed to extend the lifetime of sensors. In Sun (2008) is presented a query optimization method based on user-specified accuracy item for wireless sensor networks. When issuing a query, user may specify a value/time accuracy constraint according to which an optimized query plan can be created to minimize the energy consumption. At each single sensor node, instead of direct delivery of each reading, algorithms are proposed to reduce both data sensing and data transmission.

In Rosemark et al. (2007), a query optimizer for sensor networks whose key component is a cost-based analysis to estimate the overall energy consumption associated with a given query plan over the lifetime of a query is presented. By utilizing this estimation technique, the query optimizer compares the desirability of several candidate plans for a given query and chooses a plan that is likely to utilize the least energy for execution. Simulation results show that the query plan chosen by such query optimizer consumes significantly less energy than the state-of-the-art query processing engine for sensor networks. In Alwadi and Chetty (2015), an energy-efficient data mining approach to characterize and classify large-scale physical environments based on a combination of feature selection and single mode/ensemble classifier techniques is introduced.

By approaching the complexity of WSN/SN with a data mining formulation, where each sensor node is equivalent to an attribute or a feature of a data set, and all the sensor nodes together forming the WSN/SN setup equivalent to multiple features or attributes of the data set, it is possible to use powerful feature selection, dimensionality reduction, and learning classifier algorithms from data mining field and come up with an energy-efficient automatic classification system. Here, minimizing the number of sensors for energy-efficient control is very similar to minimizing the number of features with an optimal feature selection scheme for the data mining problem.

Conclusion and Future Direction

This chapter presented a survey of some of the most representative initiatives that have been done in the field of energy-efficient big data analysis. On top of the current approaches described above, there is a need for new methods and concepts in order to optimize the processing of big data analysis and improve the associated energy efficiency. Three main promising topics can be then considered:

- approximate computing
- quantum technologies
- low-energy processors

In light of the need for new innovations to sustain these improvements, approximate computing plays a key role. Approximate computing has recently emerged as a promising approach to energy-efficient design of digital systems. It relies on the ability of many systems and applications to tolerate some loss of quality or optimality in the computed result. By relaxing the need for fully precise or completely deterministic operations, approximate computing techniques allow substantially improved energy efficiency. The objective here is not to describe this concept in detail; nevertheless the main research topics can be mentioned:

- Approximate circuits: Components like approximate adders, multipliers, and other logical circuits can significantly reduce hardware overhead. For example, an approximate multi-bit adder can ignore the carry chain and, thus, allow all its sub-adders to perform additional operation in parallel.
- Approximate storage: Instead of storing data values exactly, they can be stored approximately, or another method is to accept less reliable memory. In general, any error detection and correction mechanisms should be disabled. Software-level approximation: There are several ways to approximate at software level like memorization, iterations of loop deletion, some tasks skipping, etc.

- Approximate system: Here, different subsystems of the system such as the processor, memory, sensor, and communication modules can be approximated to obtain a much better than the one reached at sub-system level.

Quantum computing studies computation systems that make direct use of quantum mechanical phenomena, such as superposition and entanglement, to perform operations on data. As of 2017, the development of actual quantum computers is still in its infancy, but experiments have been carried out in which quantum computational operations were executed on a very small number of quantum bits. Both practical and theoretical research continues, and large-scale quantum computers would theoretically be able to solve certain problems much more quickly than any classical computers that use even the best currently known algorithms. That implies a more efficient way to compute and will lead to a better efficiency.

Finally, low-energy processors will also play an important role in the energy efficiency and optimization battle. This domain helps design CPUs that perform tasks efficiently without overheating which is a major consideration of nearly all CPU manufacturers to date. Some CPU implementations use very little power; CPUs in mobile phones often use just a few watts of electricity, while some microcontrollers used in embedded systems may consume only a few milliwatts or less. In comparison, CPUs in general-purpose personal computers, such as desktops and laptops, dissipate significantly more power because of their higher complexity and speed. For all the manufacturers, the final objective is so to reduce the power consumption. This can be done in several ways, from voltage, frequency, capacitance reduction, or recycling some of the energy stored in the capacitors rather than dissipating it as heat in transistors.

As a highlight with these three examples, diminishing benefits from technology scaling have pushed designers to look for new sources of computing efficiency. These are part of our near future; nevertheless, as energy efficiency is becoming a major challenge for the computing

industry, progresses in the domain will continue and new concepts will emerge.

Cross-References

- [Big Data Analysis for Smart City Applications](#)
- [Big Data Analysis Techniques](#)

References

- Alsalah W, Akl SG, Hassanein HS (2005) Energy-aware task scheduling: towards enabling mobile computing over MANETs. In: IPDPS'05, pp 242a
- Alwadi M, Chetty G (2015) Energy efficient data mining scheme for high dimensional data. Procedia Comput Sci 46:483–490
- Aydin H, Melhem R, Moss D, Mejia-Alvarez P (2004) Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 53(5):584–600
- Bhargava R, Kargupta H, Powers M. (2003) Energy consumption in data analysis for on-board and distributed applications. In: ICML'03
- Bianchini R, Rajanony R (2004) Power and energy management for server systems. Computer 37(11):68–76
- Catena M, Tonello N (2017) Energy-efficient query processing in web search engines. Trans Knowl Data Eng 29:1412–1425
- Comito C, Talia D (2014) Energy-aware clustering of ubiquitous devices for collaborative mobile applications. In: Proceeding of MobiCASE, pp 133–142
- Comito C, Talia D (2017) Energy consumption of data mining algorithms on mobile phones: evaluation and prediction. In: Pervasive and mobile computing, vol 42, pp 248–264
- Comito C, Talia D, Trunfio P (2011) An energy-aware clustering scheme for mobile applications. In: IEEE Scalcom'11, pp 15–22
- Comito C, Talia D, Trunfio P (2012) An energy aware framework for mobile data mining, chapt. 23. In: Energy efficient distributed computing systems. Wiley-IEEE Computer Society Press, New Jersey
- Comito C, Falcone D, Talia D, Trunfio P (2017) Energy-aware task allocation for small devices in wireless networks. Concurrency Comput Pract Exp 29(1): 1–24
- Guo B, Yu J, Liao B, Yang D, Lu L (2017) A green framework for DBMS based on energy-aware query optimization and energy-efficient query processing. J Netw Comput Appl 84:118–130
- Kargupta H, Park B, Pitties S, Liu L, Kushraj D, Sarkar K (2002) Mobimine: monitoring the stock market from a PDA. ACM SIGKDD Explor 3(2):37–46
- Kargupta H, Bhargava R, Liu K, Powers M, Blair P, Bushra S, Dull J (2003) VEDAS: a mobile and distributed data stream mining system for RealTime vehicle monitoring. In: SIAM data mining conference
- Lang W, Patel JM (2009) Towards Eco-friendly database management systems. CoRR, vol. abs/0909.1767
- Lang W, Kandhan R, Patel JM (2011) Rethinking query processing for energy efficiency: slowing down to win the race. Computer Sciences Department, University of Wisconsin, Madison
- Lefurgy C, Rajamani K, Rawson F, Felter W, Kistler M, Keller T (2003) Energy management for commercial servers. Computer 36(12):39–48
- Li K, Kumpf R, Horton P, Anderson T (1994) A quantitative analysis of disk driver power management in portable computers. In: USENIX conference, pp 279–292
- Li Z, Wang C, Xu R (2001) Computation offloading to save energy on handheld devices: a partition scheme. In: ACM international conference compilers, architecture, and synthesis for embedded systems, pp 238–246
- Liu L, Wang H, Liu X, Jin X, He W, Wang Q, Chen Y (2009) GreenCloud: a new architecture for green data center. In: 6th international conference on autonomic computing and communications, pp 29–38
- Luo J, Jha NK (2000) Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In: ICCAD
- Mohapatra S, Venkatasubramanian N (2003) PARM: power aware reconfigurable middleware. In: 23rd international conference on distributed computing systems, pp 312–319
- Petracci V, Loques O, Niteroi B, Mossé D (2009) Dynamic configuration support for power-aware virtualized server clusters. In: 21th Euromicro conference on real-time systems
- Rosemark R, Lee WC, Urgaonkar B (2007) Optimizing energy-efficient query processing in wireless sensor networks. In: International conference on mobile data management, pp 24–29
- Roukh A, Bellatreche L, Ordóñez C (2016) EnerQuery: energy-aware query processing. In: Proceeding of the 25th ACM CIKM conference, pp 2465–2468
- Rudenko A, Reiher P, Popek GJ, Kuenneng GH (1998) Saving portable computer battery power through remote process execution. SIGMOBILE Mob Comput Commun Rev 2(1):19–26
- Seth K, Anantaraman A, Mueller F, Rotenberg E (2003) FAST: frequency-aware static timing analysis. In: IEEE RTSS, pp 40–51
- Sun JZ (2008) An energy-efficient query processing algorithm for wireless sensor networks. In: Sandnes FE, Zhang Y, Rong C, Yang LT, Ma J (eds) Ubiquitous intelligence and computing
- Verma A, Ahuja P, Neogi A (2008) Power-aware dynamic placement of HPC applications. In: International conference on supercomputing, pp 175–184
- Wang F, Helian N, Guo Y, Jin H (2003) A distributed and mobile data mining system. In: Proceeding of the international conference on parallel and distributed computing, applications and technologies

- Weiser M, Welch B, Demers A, Shenker S (1996) Scheduling for reduced CPU energy. In: Mobile computing. Springer, Boston, pp 449–471
- Yang J, Mo T, Lim L, Sattler KU, Misra A (2013) Energy-efficient collaborative query processing framework for mobile sensing services. In: IEEE 14th international conference on mobile data management, pp 147–156
- Zhang Y, Hu X, Chen D (2002) Task scheduling and voltage selection for energy minimization. In: DAC'02, pp 183–188
- Zhuo J, Chakrabarti C (2005) An efficient dynamic task scheduling algorithm for battery powered DVS systems. In: ASP-DAC'05, pp 846–849

Energy Implications of Big Data

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

[Energy wall](#); [Energy-efficient data storage and processing](#); [Green big data](#)

Definitions

Data storage, communication, and processing consume energy, and big data requires a correspondingly big energy budget, necessitating more attention and effort to ensure energy efficiency.

Overview

Until fairly recently, developments in the field of computer architecture (Parhami 2005) were focused around computation running time and hardware complexity and how the two can be traded off against each other in various designs. With the advent of mobile devices and terascale, petascale, and, soon, exascale computing, energy consumption emerged as a major design

factor that overshadowed the older concerns to some extent. Key driving forces for research into energy efficiency were battery life in compact mobile devices with smallish batteries, energy costs for operators of supercomputer centers, and the difficulties of heat dissipation in both mobile devices and mainframe/data-center installations. We are thus motivated to seek extreme energy economy measures to make the processing of large data sets feasible within power budgets that are practical for both cloud data centers and mobile devices used at the cloud's edges.

Energy-Efficiency Trends

Early digital computers of the 1940s performed about one operation per Watt-hour (Wh) of energy consumed. Those machines, which resembled factory equipment, were bulky, unreliable, and user-unfriendly, problems that overshadowed their energy inefficiency. The exponential decline of the per-computation energy requirement, or exponential rise of the number of operations performed per unit of energy (Denning and Lewis 2017), over the decades is known as Koomey's law (Koomey et al. 2011). The exponential rise of computational capability per kWh of energy is depicted in Fig. 1.

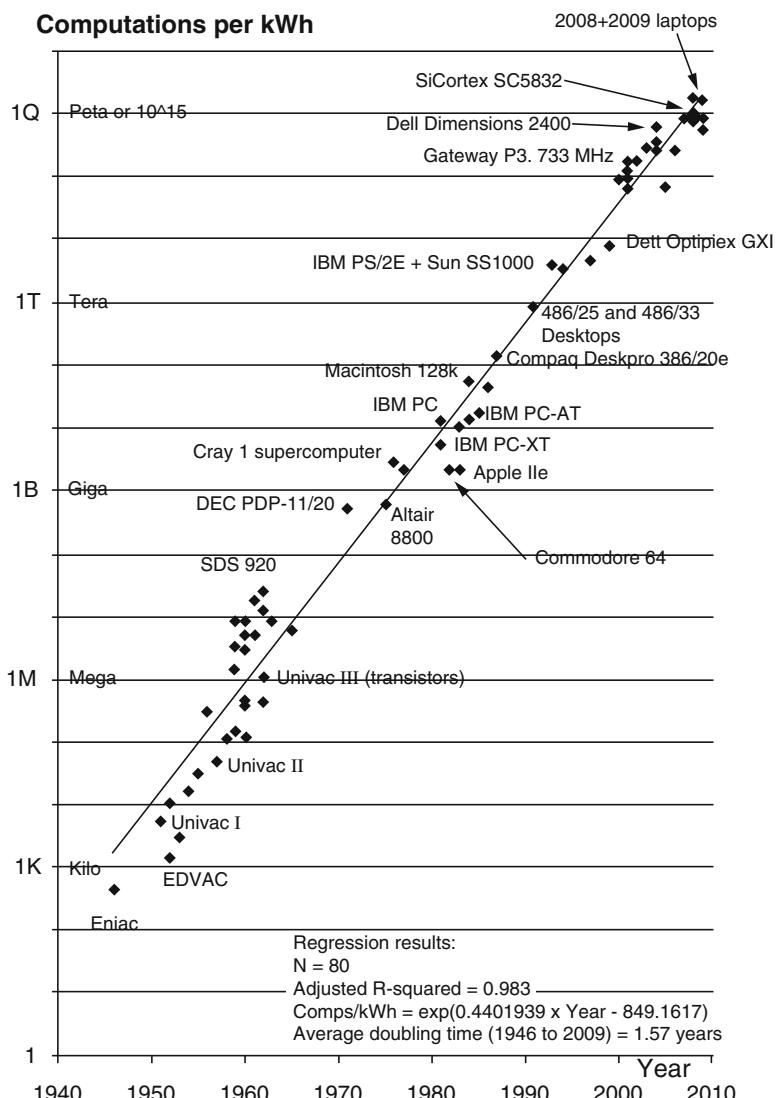
A simple back-of-the-envelope calculation based on the expected 10^{25} bytes of data produced per day by 2020 leads to the conclusion that even at the optimistic 10^{18} computations per kWh, energy requirements of big-data processing will be prohibitive.

Causes of Energy Dissipation

As energy efficiency of mainstream processors that control servers, desktops, laptops, and smartphones has improved, research into ultralow-power digital circuits has paved the way for even greater reduction in energy requirements per computation. Physical laws dictate that any nonlinear transformation of the kinds performed by standard AND/OR logic

Energy Implications of Big Data, Fig. 1

Trends in energy efficiency, expressed as computations per kWh (Koomey et al. 2011)



gates must necessarily expend energy, and a lower bound for this energy is known to be on the order of kT , where $k \approx 1.38 \times 10^{-23}$ is the Boltzmann constant and T is the operating temperature in Kelvin (Landauer 1961). This lower bound has come to be known as Landauer's principle (Bennett 2003). Although, in principle, this minimum should be reachable, current computing devices are a great deal less efficient in the amount of energy they dissipate.

The energy used by a computing device is the product of average power drawn and the computation time. This relationship suggests two

ways of reducing the energy consumption: Using lower-power technologies (e.g., Jaberipur et al. 2018) and using faster algorithms. Use of faster algorithms is, of course, highly desirable, as it has performance implications as well. So, there is a great deal of ongoing research to find ever-faster algorithms for computational problems of interest. Power consumption per device has been going down due to Dennard scaling, a law devised in 1974 that maintains the relative constancy of MOSFET power density, as transistors get smaller, a trend predicted by Moore's law (Brock and Moore 2006; McMenamin 2013).

The power dissipated by computing devices consists of static and dynamic components. Static power is the power used by devices that are in the off state and ideally should not be drawing any power; however, leakage and other factors lead to some power waste, which is increasing in significance as we use smaller and denser circuits. Dynamic power (Benini et al. 2001) is proportional to the operating voltage squared, the operating frequency, circuit capacitance, and activity (prevalence of signal-value changes). Low-voltage circuits use much less power, but the closer we get to the threshold voltage, the slower and less reliable circuit operation becomes (Kaul et al. 2012). Thus, there is a limit to power savings by reducing the operating voltage.

Reasons for energy waste include glitching (Devadas et al. 1992), unnecessary signal-value transitions that can be avoided by suitable encodings (e.g., Musoll et al. 1998), and not clocking the circuitry that is not involved in performing useful computations (Wu et al. 2000). Practical notions in this regard include energy-proportional computing, that is, the desirable property that the energy used is commensurate with the amount of useful work performed (Barroso and Holzle 2007), and dynamic frequency AND/OR voltage scaling (Nakai et al. 2005) in an effort to save energy when application requirements do not demand the highest performance level.

A data center's energy efficiency is influenced by three factors attributed to the facility, server conversion efficiency, and efficiency at the level of circuits and devices (Barroso et al. 2013). Only about a third of the energy is used for productive computation (Holzle 2017). Generally speaking, large-scale installations provide more opportunities for energy fine-tuning, thus underlining the importance of data centers and cloud-computing in handling big-data loads. More on this later.

Near-Threshold Multi-core Computing

Parallel processing is nearly a necessity for handling of large data volumes, on performance

grounds. It is also helpful for reducing power consumption. Using m processors/cores, each with $1/m$ the speed, is more energy-efficient than using a single high-performance processor. Thus, if an application offers enough parallelism to use multiple processors/cores efficiently, it will lead to higher performance AND/OR lower energy consumption. Beginning with dual-core chips in the mid-2000s, multi-core chips (Gepner and Kowalić 2006) have allowed performance scaling to continue unabated, despite Moore's Law, in its original formulation, becoming invalid (Mack 2011).

While use of a large number of simple cores carries energy-efficiency benefits, one must also pay attention to the effect of inherently sequential parts of a computation which give rise to speedup limits predicted by Amdahl's law (Parhami 1999). Perhaps a judicious mix of powerful or "brawny" and simple or "wimpy" cores (Holzle 2010) in a multi-core chip, possibly also including application-specific performance boosters (Hardavellas et al. 2011), can help mitigate this problem.

The ultimate in energy efficiency is achieved when the processors/cores operate very near the threshold voltage (Dreslinski et al. 2010; Gautschi 2017; Hubner and Silano 2016). As operating voltage is scaled down, performance is reduced nearly linearly, whereas active energy dissipation goes down quadratically. Therefore, performance per unit of energy improves. When the static or leakage power is taken into account, the savings become less pronounced but still significant. Near-threshold computing has pitfalls in terms of computation stability and system reliability, so there is a sensitive trade-off to be performed. There is also the issue of giving up too much in sequential performance in the hopes of regaining some of it via parallel processing (Mudge and Holzle 2010).

Memory/Storage Energy Requirements

Dynamic random-access memory (DRAM) chips account for a significant fraction of the power

used in a computer's electronic parts. Static RAM (SRAM) is even more energy-intensive, but because SRAM sizes tend to be relatively much smaller, they do not contribute as much to the energy budget of a system. Precise modeling of DRAM energy consumption has been attempted in an effort to determine the exact share of power used; importance of power dissipation within the memory cells vs. peripheral circuitry for decoding, access, and content refreshing; and expected changes as DRAM technology is scaled down and goes through other generational changes (Vogelsang 2010). These methods will guide energy reduction in main memories in the near future. For the longer term, a variety of main memory technologies, some of which enable inherently lower-power operation, are being investigated (Xie 2011).

Disk memories are both slow and energy-intensive but have the advantage of permanence, low cost per bit, and nonvolatility. For small to modest memory sizes, replacement of disks with solid-state memories is feasible, but for very large data volumes, such a replacement is still impractically expensive. So, in the short term, we have to live with disk memory shortcomings. Even though a single disk unit has a long mean time to failure (MTTF), when many thousands of disks are involved in a storage warehouse, failures are inevitable. Modern data storage facilities make use of redundant arrays of independent disks, RAID for short (Schroeder and Gibson 2007), to mitigate the slowness and low reliability of disk units. A variety of methods have been devised to make disk memories and attendant units, such as disk caches and RAID controllers, that make a disk array look like a single disk and reconstruct damaged or lost data upon disk failures, more energy-efficient (Pinheiro and Bianchini 2014).

As storage devices are increasingly networked to gain the reliability, availability, and scale benefits of distributed access, communication overhead and its attendant energy requirements must be factored in when comparing different organizations with regard to energy efficiency.

Communication Energy Requirements

Like computation and storage, communication also dissipates energy. In fact, one of the most challenging aspects of managing big-data applications is to decide how the energy budget should be allocated to the three aspects of computing, storage, and communication. Data replication often reduces communication costs during access but may impose a nontrivial overhead for keeping replicas consistent, and storage of previously computed results may obviate the need for re-computation.

Reducing communication costs has been studied extensively in connection with energy-limited mobile and sensor networks (Heinzelman et al. 2000), but many of the techniques are applicable more broadly. For example, expending local computation to aggregating data before transmitting a smaller volume of data is a generally useful method (Krishnamachari et al. 2002).

Given their current prevalence and future importance, energy efficiency consideration for on-chip networks and those used in data centers are of paramount importance. Networks-on-chip (Benini and De Micheli 2002; Pande et al. 2005) are key components of modern multi-core chips, and their importance will increase as the number of cores is scaled up. Data-center networks have been subjects of numerous studies, given their role in the energy costs of a typical data center (Hammadi and Mhamdi 2014). One example consists of the proposal for energy-proportional data-center networks (Abts et al. 2010).

Beyond networks-on-chip and data-center networks, we must also be concerned with energy efficiency in broad-area networks, the Internet in particular (Bolla et al. 2011). Whereas the introduction of low-power techniques in the design of routers and other electronic units is necessary for energy-efficient networking, it is not enough. We also need energy savings at the physical network layer and in the algorithms and protocols used for computation, routing, resource management, testing, and fault tolerance.

Energy Considerations in the Cloud

The cloud consists of computational, storage, and communication resources. Yet, to achieve energy efficiency in the cloud, we must go beyond separate energy optimizations for the aforementioned components. Such optimizations are challenging, given the scale of the cloud (Assuncao et al. 2015; Feller et al. 2015). As of 2017, servers collectively use about 200 TWh of energy, which is comparable to energy use in all of Mexico. Google alone uses as much energy as the city of San Francisco. As large as this value is, it is dwarfed compared with the energy required by users' laptops or other computing equipment at the edges of the cloud.

As for the energy used in data-center installations, it consisted until fairly recently of three nearly equal parts devoted to mechanical cooling, IT equipment, and everything else (lighting, backup power supply, etc.). So, the energy used for the actual computation was multiplied by a factor of 3.0, implying a 200% overhead. More efficient modern data centers reduced this factor to 1.8, for an overhead of 80%. Now, we can go as low as 10% overhead through a variety of energy-saving schemes, including the application of machine learning to adjust a building's cooling strategy based on information about the applicable parameters (Holzle 2017).

Servers have undergone similar efficiency improvements. Earlier, some 50% of energy went to waste, even before power got to the actual circuits. By eliminating this waste, we are now at about 10% overhead relative to the actual energy used by the circuits. The circuit energy has been going down by 20% per year in recent years (post-Moore's Law era). Factors leading to this reduction are smaller circuits, clock-gating (disabling the parts of the circuits not in use, so that they don't draw energy), frequency scaling, and specialization (tailoring the circuits to computations). In the latter domain, Google's hardware optimized for machine learning uses 0.2 MW of power, compared with 2.5 MW needed by a general-purpose supercomputer doing the same job (Holzle 2017).

As a whole, the IT industry uses about 2% of the world's energy, which is of the same order as the amount used by airlines. Because modern data centers are way more efficient than local server installations, moving to the cloud will reduce the energy consumption associated with computations by some 87%. Operating data centers with exclusive use of renewable energy is now possible, which constitutes a major benefit. The lower hardware redundancy needed to ensure reliable operation also saves energy. For example, Gmail uses 1% redundancy in hardware resources, whereas a typical local e-mail server installation needs at least duplication to avoid service disruptions. This makes the user-side energy consumption even more important. Fortunately, with the move away from desktops and laptops to tablets and smartphones, user-side energy consumption is also plummeting.

Future Directions

Work is proceeding in improving energy efficiency in all aspects of the big-data environment discussed in the preceding sections. At the circuit and chip levels, ultralow-power technologies are being devised and evaluated. Examples are found in the fields of optical, biologically inspired, adiabatic, and reversible circuits. The ultimate goal might be doing away with batteries and other power sources altogether (Eisenberg 2010), for a large fraction of the big-data ecosystem. Significant reduction in the dissipated energy will also obviate the need for complex cooling strategies and their associated hardware and software costs (Kaushik and Nahrstedt 2012).

The future of big data is closely tied to the future of cloud computing, as the economy of scale provided by the cloud is necessary for the successful deployment of big-data applications (Hashem et al. 2015; Wu et al. 2013). A promising direction in energy monitoring and optimization, with no need for human intervention, is the use of machine learning strategies. Such techniques have already been applied to power management at the circuit level (Dhiman and Rosing

2006) and to data-center climate control systems (Holzle 2017), but a great deal more can be done.

Cross-References

- [Big Data and Exascale Computing](#)
- [Computer Architecture for Big Data](#)
- [Emerging Hardware Technologies](#)
- [Parallel Processing with Big Data](#)

References

- Abts D, Marty MR, Wells PM, Klausler P, Liu H (2010) Energy proportional datacenter networks. ACM SIGARCH Comput Archit News 38(3):338–347
- Assuncao MD, Calheiros RN, Bianchi S, Netto MAS, Buyya R (2015) Big data computing and clouds: trends and future directions. J Parallel Distrib Comput 79: 3–15
- Barroso LA, Holzle U (2007) The case for energy-proportional computing. IEEE Comput 40(12):33–37
- Barroso LA, Clidara J, Holzle U (2013) The data center as a computer: an introduction to the design of warehouse-scale machines. Synth Lect Comput Archit 8(3):1–154
- Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. IEEE Comput 35(1):70–78
- Benini L, De Micheli G, Macii E (2001) Designing low-power circuits: practical recipes. IEEE Circuits Syst Mag 1(1):6–25
- Bennett CH (2003) Notes on Landauer's principle, reversible computation and Maxwell's Demon. Stud Hist Philos Mod Phys 34(3):501–510
- Bolla R, Bruschi R, Davoli F, Cucchietti F (2011) Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures. IEEE Commun Surv Tutorials 13(2):223–244
- Brock DC, Moore GE (eds) (2006) Understanding Moore's law: four decades of innovation. Chemical Heritage Foundation, London
- Denning PJ, Lewis YG (2017) Exponential laws of computing growth. Commun ACM 60(1):54–65
- Devadas S, Keutzer K, White J (1992) Estimation of power dissipation in CMOS combinational circuits using boolean function manipulation. IEEE Trans Comput Aided Des Integr Circuits Syst 11(3):373–383
- Dhiman G, Rosing TS (2006) Dynamic power management using machine learning. In: Proceedings of IEEE/ACM international conference on computer-aided design, pp 747–754
- Dreslinski RG, Wieckowski M, Blaauw D, Sylvester D, Mudge T (2010) Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. Proc IEEE 98(2):253–266
- Eisenberg A (2010) Bye-Bye batteries: radio waves as a low-power source. New York Times, July 18, p BU3. <http://www.nytimes.com/2010/07/18/business/18novel.html>
- Feller E, Ramakrishnan L, Morin C (2015) Performance and energy efficiency of big data applications in cloud environments: a Hadoop case study. J Parallel Distrib Comput 79:80–89
- Gautschi M (2017) Design of energy-efficient processing elements for near-threshold parallel computing, doctoral thesis, ETH Zurich
- Gepner P, Kowalik MF (2006) Multi-core processors: new way to achieve high system performance. In: Proceedings of IEEE international symposium on parallel computing in electrical engineering, pp 9–13
- Hammadi A, Mhamdi L (2014) A survey on architectures and energy efficiency in data center networks. Comput Commun 40:1–21
- Hardavellas N, Ferdman M, Falsafi B, Ailamaki A (2011) Toward dark silicon in servers. IEEE Micro 31(4):6–15
- Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Ullah Khan S (2015) The rise of 'Big Data' on cloud computing: review and open research issues. Inf Syst 47:98–115
- Heinzelman WR, Chandrakasan A, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of 33rd Hawaii international conference on information sciences, IEEE, pp 10
- Holzle U (2010) Brawny cores still beat wimpy cores, most of the time. IEEE Micro 30(4):23–24
- Holzle U (2017) Advances in energy efficiency through cloud and ML, energy leadership lecture, University of California, Santa Barbara
- Hubner M, Silano C (eds) (2016) Near threshold computing: technology, methods, and applications. Springer, Cham
- Jaberipur G, Parhami B, Abedi D (2018) Adapting computer arithmetic structures to sustainable supercomputing in low-power, majority-logic nanotechnologies. IEEE Trans Sustain Comput. <https://doi.org/10.1109/TSUSC.2018.2811181>
- Kaul H, Anders M, Hsu S, Agarwal A, Krishnamurthy R, Borkar S (2012) Near-threshold voltage (NTV) design – opportunities and challenges In: Proceedings of 49th ACM/EDAC/IEEE design automation conference, pp 1149–1154
- Kaushik RT, Nahrstedt K (2012) T*: a data-centric cooling energy costs reduction approach for big data analytics cloud. In: Proceedings of international conference on high performance computing, networking, storage and analysis, pp 1–11
- Koomey JG, Berard S, Sanchez M, Wong H (2011) Implications of historical trends in the electrical efficiency of computing. IEEE Ann Hist Comput 33(3):46–54
- Krishnamachari L, Estrin D, Wicker S (2002) The impact of data aggregation in wireless sensor networks. In: Proceedings of 22nd international conference on distributed computing systems, pp 575–578

- Landauer R (1961) Irreversibility and heat generation in the computing process. *IBM J Res Dev* 5(3):183–191
- Mack CA (2011) Fifty years of Moore's law. *IEEE Trans Semicond Manuf* 24(2):202–207
- McMenamin A (2013) The end of Dennard scaling, online document. <http://cartesianproduct.wordpress.com/2013/04/15/the-end-of-dennard-scaling/>. Accessed 20 Feb 2018
- Mudge T, Holzle U (2010) Challenges and opportunities for extremely energy-efficient processors. *IEEE Micro* 30(4):20–24
- Musoll E, Lang T, Cortadella J (1998) Working-zone encoding for reducing the energy in microprocessor address buses. *IEEE Trans VLSI Syst* 6(4):568–572
- Nakai M, Akui S, Seno K, Meguro T, Seki T, Kondo T, Hashiguchi A, Kawahara H, Kumano K, Shimura M (2005) Dynamic voltage and frequency management for a low-power embedded microprocessor. *IEEE J Solid State Circuits* 40(1):28–35
- Pande PP, Grecu C, Jones M, Ivanov A, Saleh R (2005) Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans Comput* 54(8):1025–1040
- Parhami B (1999) Chapter 7, sorting networks. In: *Introduction to parallel processing: algorithms and architectures*. Plenum Press, Plenum, New York, pp 129–147
- Parhami B (2005) Computer architecture: from microprocessors to supercomputers. Oxford University Press, New York
- Pinheiro E, Bianchini R (2014) Energy conservation techniques for disk array-based servers. In: Proceedings of ACM international conference on supercomputing, pp 369–379
- Schroeder B, Gibson GA (2007) Understanding disk failures rates: what does an MTTF of 1,000,000 hours mean to you? *ACM Trans Storage* 3(3):Article 8
- Vogelsang T (2010) Understanding the energy consumption of dynamic random access memories. In: Proceedings of 43rd IEEE/ACM international symposium on microarchitecture, pp 363–374
- Wu L, Barker RJ, Kim MA, Ross KA (2013) Navigating big data with high-throughput, energy-efficient data partitioning. *ACM SIGARCH Comput Archit News* 41(3):249–260
- Wu Q, Pedram M, Wu X (2000) Clock-gating and its application to low power design of sequential circuits. *IEEE Trans Circuits Syst I* 47(3):415–420
- Xie Y (2011) Modeling, architecture, and applications for emerging memory technologies. *IEEE Des Test Comput* 28(1):44–51

Energy Wall

- Energy Implications of Big Data

Energy-Efficiency Benchmarking

- Energy Benchmarking

Energy-Efficient Data Storage and Processing

- Energy Implications of Big Data

Entity Resolution

- Record Linkage

ETL

Christian Thomsen
Department of Computer Science, Aalborg University, Aalborg, Denmark

Synonyms

- ELT; Extract-Transform-Load

Definitions

ETL is short for Extract-Transform-Load. The ETL process extracts data from operational source systems, transforms the data, and loads the data into a target. The transformations to perform on the data can involve a plethora of different activities, e.g., filtering, normalization or de-normalization to a desired form, joins, conversion, and cleansing to remove bad or dirty data. In the ELT variant, the data is extracted from the source systems, loaded in its raw form into the target, and then transformed.

Overview

The term ETL process has traditionally been used for a process that populates a data warehouse (DW) managed by a relational database management system (RDBMS). As pointed out by Simitsis and Vassiliadis (2017), the basic concept of populating a data store with data reshaped from another data store is, however, older than data warehousing. The ETL process can be hand-coded or made with a designated ETL tool where the developer typically uses graphical constructs to define the ETL process. With the emergence of cheap technologies for capturing and processing Big Data, the requirements to ETL tools have increased. First, new types of data sources have appeared, for example, NoSQL systems, files in the Hadoop Distributed File System (HDFS), and streams from Apache Kafka. Second, the target to load data into may be something different from a DW managed by an RDBMS. Organizations with very big amounts of data are now using many clusters of commodity computers (or clouds in case the machines are virtual) for data processing and data analytics. In such a case, the ETL process, for example, has to load the data into HDFS files.

Both commercial and open source ETL tools now typically provide support for Big Data sources. It varies from tool to tool exactly which types of sources are supported, but reading data from HDFS files is very widely supported. This kind of support thus enables extracting existing data from a Big Data system. Querying and transformation of such data are also supported by varying degrees. Basic support includes the execution of a predefined task on a Big Data processing platform. Such a task could, for example, be a HiveQL query on Hive, a MapReduce job on Hadoop, or a file operation on HDFS. More advanced ETL tools in addition support that the developer uses the ETL environment and its transformations to define an ETL flow which then automatically is translated into a native program (e.g., Spark or MapReduce) that can be executed on a Big Data

processing platform. The same ETL flow can then be used both for testing and development on a stand-alone machine with little data and for production on a cluster with large amounts of data. Tools supporting this are typically based on the visual paradigm where the developer connects boxes representing transformations with lines representing dataflow. Much development of ETL flows for Big Data is, however, done by manual programming of Spark programs or the like. This allows the developer to easily integrate with the existing tool chain and to use the quickly evolving functionality of the data processing platform. Databricks report that the most widely used language for ETL programming on Spark is Python (No matter if visual or code-based ETL flow definitions are used, the use of a massively parallel system like Spark enables processing of larger and more complex datasets than can be handled in more traditional setups with a single ETL server. The transformed and cleaned data can then be loaded into a traditional DW managed by an RDBMS, or it can be stored directly in files (e.g., in ORC or Parquet format) in HDFS to enable analysis from tools like Hive and Spark. In the latter case, it is often necessary to also use data residing in a relational database. To support the need for efficient bulk transfers between Hadoop and relational databases, the open source and command-line-based tool Sqoop (<http://sqoop.apache.org/>) was created.

With the advent of cloud computing platforms available on demand (such as Amazon Web Services and Google Cloud Platform) and data warehousing solutions for them (such as Amazon Redshift (Gupta et al. 2015), Snowflake (Dageville et al. 2016), and Google BigQuery (Tigani and Naidu 2014)), on-demand ETL services have also appeared recently. Examples include Matillion, Attunity Compose, and AWS Glue. With such services, the user pays for the use of the ETL software and the hardware it is running on based on how long the software is running. This thus makes it possible to start using the ETL services with very low upfront payment and to scale to larger hardware when needed.

Key Research Findings

Cloud computing gained momentum in the 2000s, and this opened possibilities for collecting and analyzing big datasets. In 2004, Google published a paper about their MapReduce framework (Dean and Ghemawat 2004) which soon became very popular. It offered an easy way to make a highly scalable and fault-tolerant program for a cluster of hundreds or thousands of computers. The developer only had to define two functions – *map* which processes different parts of the input in parallel and produces intermediate key/value pairs and *reduce* which for a given intermediate key and all its values produces a result. Open source versions of MapReduce and its distributed file system were implemented in Apache’s Hadoop and HDFS, respectively, as well as in other frameworks. MapReduce was not only used for analysis of data but also preprocessing of data such as cleansing and transformations. IBM reported that many customers used MapReduce to preprocess data before loading it into DW managed by the DB2 DBMS (Özcan et al. 2011). While MapReduce conceptually is simple, there are still many technical aspects to pay attention to when using it and the developer uses a general-purpose programming language (such as Java for Apache Hadoop) to express what to do. Further high-level functionality (such as joins) is not built-in. It can thus be tedious and error prone to write MapReduce programs. As a remedy, Facebook proposed Hive (Thusoo et al. 2009) where the user can use the high-level SQL-like HiveQL language. HiveQL queries are then compiled into MapReduce jobs by Hive’s query compiler. Hive also supports so-called external tables with data written to files by other applications and can be extended to read other input formats. It can also use user-defined functions (UDFs) written in Java. It is thus possible to use Hive to do much ETL/ELT processing of data.

Another abstraction on MapReduce is Pig (Olston et al. 2008) where the developer specifies sequences of transformations in a high-level procedural language called Pig Latin. Pig is made for parallel data processing and can operate on rela-

tional, nested, and unstructured data. It has built-in functionality for joining, grouping, ordering, filtering, etc. and can also be extended by UDFs. Pig thus also became popular for transforming and cleaning data in ETL flows. Pig can now execute on MapReduce as well as other platforms. Sawzall (Pike et al. 2005) is a similar tool created by Google.

ETLMR (Liu et al. 2013) was proposed as a high-level dimensional ETL tool which loads data into a traditional relational database. It executes on MapReduce but hides the complexity of MapReduce. The developer writes Python code to specify sources, targets, and transformations. ETLMR has built-in DW-specific support for handling dimension tables and fact tables. The developer can thus, for example, find changes and possibly insert a new version of a member in a slowly changing dimension (Kimball 2008) with a single method call. With ETLMR, dimension processing is always done first followed by fact processing. The developer can choose between different dimension processing schemes deciding how dimension data is partitioned between different reducers. CloudETL (Liu et al. 2014) has a similar approach but is aimed at loading data into Hive. The developer defines the ETL flow by high-level constructs in Java, and the system will then handle the parallelization. It remedies that it is hard to do dimensional ETL processing in Hive and Pig which are made for batch processing and not individual look-ups or inserts and which do not support UPDATEs such that slowly changing dimensions are very difficult to implement, although they are widely used in traditional DWs. For big dimensions (i.e., dimensions with many members), CloudETL also supports a map-only job where data locality in the distributed file system is exploited to avoid the need of shuffling of data from mappers to reducers.

Many other cluster computing frameworks have been proposed after MapReduce. Recently, Spark (Zaharia et al. 2012) has become very popular, and it is now a very active Apache project. Spark can keep datasets in memory as the so-called resilient distributed datasets (RDDs) between jobs and avoids expensive I/O which MapReduce has to do. Spark does not

use MapReduce but is closely integrated with Hadoop and works with HDFS. Spark has APIs for different languages including Scala, Java, and Python and gives the developer much flexibility in creating transformations. With the Spark SQL module (Armbrust et al. 2015), it is possible to manipulate data using both procedural and relational APIs. The relational API, however, enables richer optimizations and works both with Spark's RDDs and external data sources. To enable better optimizations, the use of the relational operations in Spark SQL builds a logical plan which first is executed when data must be output. The relational operations can be expressed in a domain-specific language. To perform the optimizations, Spark SQL provides a relational optimizer named Catalyst which is extensible such that support for new data sources and optimization rules as well as user-defined functions and data types can be added.

With the Dataflow model, Google introduced a high-level simple programming model for scalable parallel data processing with “the ability to dial in precisely the amount of latency and correctness for any specific problem domain given the realities of the data and resources at hand” (Akidau et al. 2015). In the model, data is not viewed as anything that eventually will be complete, and the same operators can thus be used for bounded data and unbounded streaming data. The physical execution engine is abstracted away under the programming model, and different execution engines can thus be used. Apache Beam (<https://beam.apache.org/>) is an open source version of Dataflow and does also support a number of different execution engines.

BigDansen (Khayyat et al. 2015) is a data cleansing system for Big Data. The user expresses a logical plan by means of five logical operators which are UDFs allowing great expressibility. The operators are defined at the finest unit of the input data (e.g., rows for relational data and triplets for RDF) to allow highly parallel execution of the operators. BigDansen translates a logical plan into an optimized physical plan which in turn is translated into an execution plan for a specific execution engine such as MapReduce or Spark. The user does thus only specify the

UDFs in Java code and does not need expertise on the specific execution engine.

It is generally known that it is very time-consuming to transform and clean data to prepare it for analysis. Traditionally such preprocessing has been done by specialized IT staff when creating the ETL flow. In the Big Data context where new and complex data sources often must be explored and used, this is impractical. Wrangler (Kandel et al. 2011) is an interactive system where the user can be a business analyst rather than an IT professional. The user specifies transformation scripts in an interactive way by selecting data and choosing among suggested, configurable transforms and immediately review the results. When the necessary transformations have been defined, a Wrangler script can be exported to MapReduce and Python programs. Wrangler has been commercialized by the product Trifacta which also works with cloud platforms. In early 2017, the service Google Dataprep was launched to allow users to prepare data visually. It is based on Trifacta's software.

Another interesting system which is not oriented toward specialized IT staff is Data Tamer (Stonebraker et al. 2013) (commercialized as Tamr). Data Tamer uses machine learning and statistics to do schema integration and entity consolidation (i.e., deduplication). Only when needed, a human expert is asked. The human expert is a business user and not a programmer. By using automation, Data Tamer avoids the need for human specification of complex rules and can scale to a huge amount of data sources as present in the Big Data era.

Examples of Application

Organizations often have to deal with large amounts of complex and dirty data to be prepared for analysis. With the large amounts of data, dirty data is the norm, and the data is often semi-structured or unstructured. The data can, e.g., be open data from publicly available web sources, data harvested from social media, licensed commercial data, log data about page views or ad clicks, and sensor data from IoT

devices. ETL pipelines are used to cleanse such data and integrate it with existing data to enable analysis. The ETL pipelines can be created in an ETL tool but are often made of homegrown solutions exploiting a variety of other Big Data tools such as Pig, Spark, etc.

Future Directions for Research

At the time of this writing, a number of Big Data ETL tools are available. Optimization and automatic tuning of ETL pipelines for Big Data is an area for further research. There will most likely be an increasing demand for ETL tools for Big Data which are efficient and also easy to use for less technically inclined users to enable self-service. Supporting the user in discovering relevant datasets and structures of complex, dirty data is another area. As web data often changes structure, the development of self-repairing ETL flows is also an interesting direction.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Data Cleaning](#)
- ▶ [Data Deduplication](#)
- ▶ [Data Integration](#)
- ▶ [Data Wrangling](#)
- ▶ [Hadoop](#)
- ▶ [Hive](#)
- ▶ [Spark SQL](#)

References

Akida T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc VLDB Endow 8(12):1792–1803

Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A et al (2015) Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 1383–1394

- Dageville B, Cruanes T, Zukowski M, Antonov V, Avanes A, Bock J, Claybaugh J, Engovatov D, Hentschel M, Huang J, Lee AW, Motivala A, Munir AQ, Pelley S, Povinec P, Rahn G, Triantafyllis S, Unterbrunner P (2016) The snowflake elastic data warehouse. In: Proceedings of the 2016 international conference on management of data, SIGMOD'16, New York. ACM, pp 215–226. ISBN:978-1-4503-3531-7. <http://doi.acm.org/10.1145/2882903.2903741>
- Dean J, Ghemawat S (2004) Mapreduce: simplified data processing on large clusters. In: 6th symposium on operating system design and implementation (OSDI 2004), San Francisco, 6–8 Dec 2004, pp 137–150. <http://www.usenix.org/events/osdi04/tech/dean.html>
- Gupta A, Agarwal D, Tan D, Kulesza J, Pathak R, Stefan S, Srinivasan V (2015) Amazon redshift and the case for simpler data warehouses. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD'15, New York. ACM, pp 1917–1923. ISBN:978-1-4503-2758-9. <http://doi.acm.org/10.1145/2723372.2742795>
- Kandel S, Paepcke A, Hellerstein J, Heer J (2011) Wrangler: interactive visual specification of data transformation scripts. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM, pp 3363–3372
- Khayyat Z, Ilyas IF, Jindal A, Madden S, Ouzzani M, Papotti P, Quiané-Ruiz J-A, Tang N, Yin S (2015) Bigdansing: a system for big data cleansing. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 1215–1230
- Kimball R (2008) The data warehouse lifecycle toolkit. Wiley, Hoboken
- Liu X, Thomsen C, Pedersen TB (2013) Etlmr: a highly scalable dimensional ETL framework based on mapreduce. In: Hameurlain A, Küng J, Wagner RR (eds) Transactions on large-scale data-and knowledge-centered systems VIII. Springer, Heidelberg/New York, pp 1–31
- Liu X, Thomsen C, Pedersen TB (2014) Cloudetl: scalable dimensional ETL for hive. In: Proceedings of the 18th international database engineering & applications symposium. ACM, pp 195–206
- Olston C, Reed B, Srivastava U, Kumar R, Tomkins A (2008) Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, pp 1099–1110
- Özcan F, Hoa D, Beyer KS, Balmin A, Liu CJ, Li Y (2011) Emerging trends in the enterprise data analytics: connecting hadoop and db2 warehouse. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data. ACM, pp 1161–1164
- Pike R, Dorward S, Griesemer R, Quinlan S (2005) Interpreting the data: Parallel analysis with Sawzall. Sci Program 13(4):277–298
- Simitsis A, Vassiliadis P (2017) Extraction, transformation, and loading. Springer, New York, pp 1–9.

- ISBN 978-1-4899-7993-3. https://doi.org/10.1007/978-1-4899-7993-3_158-3
- Stonebraker M, Bruckner D, Ilyas IF, Beskales G, Cherniack M, Zdonik SB, Pagan A, Xu S (2013) Data curation at scale: the data tamer system. In CIDR
- Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R (2009) Hive: a warehousing solution over a map-reduce framework. Proc VLDB Endow 2(2):1626–1629
- Tigani J, Naidu S (2014) Google BigQuery analytics. Wiley, Indianapolis
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S (2012) Stoica I resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation. USENIX Association, pp 2–2

Event Log Cleaning for Business Process Analytics

Andreas Solti

Vienna University of Economics and Business,
Vienna, Austria

Synonyms

[Event log preprocessing](#); [Event log repair](#)

Definitions

Event log cleaning is a data preparation phase that turns event data into event logs to enable or improve the quality of business process analytics methods like process mining, model enrichment, and conformance checking. Event data might have to be collected from different sources and formats, filtered, transformed, and assigned to the corresponding processes and cases.

Overview

The goal of business process analytics projects is to gain insights into the execution of business processes. It can help to know which questions should be answered by the analysis. Some typical questions are *what* is done (activities), *when* is it done or how long does it take (time

stamps), in which *order* (relations), and by *whom* (resources). In contrast to traditional questionnaires, the process participants do not need to be personally asked about their perception of the process. In business process analytics, the event logs containing process execution data are directly analyzed (Dumas et al. 2013).

The quality of the underlying event logs is decisive for the success of any process analytics project. In personal interviews, the experience and insights of a person, as well as their honesty and willingness to answer questions, influence the analysis results. “Asking the data” might seem to be a less complicated task at first leading to objective analytics results. However, as experiences in process mining show, “most real-life logs tend to be incomplete, noisy, and imprecise” (Bose et al. 2013). Failing to address the data quality issues of an event log can compromise the validity of the analytics results. Rendering all the analytics efforts mute. Therefore, it is crucial to be aware and address event log quality issues before conducting any analytics. This chapter highlights typical quality issues of event logs, offers general solution strategies, and points to exemplary solution approaches available in literature.

Event Log Quality Issues

Before starting any cleaning procedure, the analysts need to understand the nature of the given event logs. To that end, a model of data quality is helpful. Different models of data quality exist in the literature that look at the dimensions of accuracy, completeness, and consistency of data, cf. (Suriadi et al. 2017, Section 2.2). To get a better understanding in the process analytics context, it is helpful to distinguish *inherent* characteristics of the business process and *system-dependent* characteristics that appear in the recorded data (International Organization for Standardization 2011).

Process Characteristics

When looking at the business process itself, the following list of challenges and solution

approaches help cleaning event logs and preparing them for analytics.

Volume. The simplest reason for large sizes of event logs is the high number of cases that are processed in large organizations. Certain log cleaning approaches perform costly operations and can take a lot of time to run.

Strategies: Divide and conquer. Stream processing.

Approaches: Scalable approaches need to be chosen for handling large event logs. At times, the available implementations of an approach are not optimized for handling huge event logs, as they require to load the event log into computer memory. In these cases, approaches with implementations that rely on divide and conquer and stream processing technologies are preferable, e.g., Leemans (2017).

Variability. Many process execution environments require a certain degree of flexibility to perform the tasks at hand. For example, doctors need to react and decide which action they need to do next based on a case by case basis. This flexibility can create a high variability in the event log, which needs to be taken into account for analytics and when cleaning the event logs (Reichert and Weber 2012). Variability can occur on the case level, that is, a lot of different paths are taken to reach one business objective. Variability can also occur in the number of different activities that can be performed.

Strategies: Divide and conquer. Clustering behavior. Focusing on frequent behavior–infrequent behavior in the event log, however, is not always irrelevant and can be separately analyzed.

Approaches: Trace clustering is proposed in Greco et al. (2006) and applied in complex and variable processes in Mans et al. (2008) to analyze the clusters separately with process mining. A more recent approach specifically addressing high variability is taking the divide and conquer approach to the limit by focusing on individual cases (Diamantini et al. 2016).

Veracity. The process of recording events in a business process can itself be subject to errors.

Examples of erroneous event log data appear due to human recording of events, due to noisy sensors tracking the progress in a process. The problem of stripping noisy and irrelevant data from event logs is one of the most important challenges in cleaning event logs for process analytics.

Strategies: Remove outliers. Repair data (with or without external knowledge). Store raw data for other analyses.

Approaches: Under the assumption that noisy data is less frequent than regular data, the events can be filtered by frequency. The concept of anomaly detection is applied to single cases in an event log in de Lima Bezerra and Wainer (2013). Here, mainly frequencies are used, as also in Conforti et al. (2017). If a process model exists, the event log can be aligned to it to identify and correct unforeseen behavior (de Leoni et al. 2015) effectively making the log fit into the model with minimum corrections.

Change. Business processes can be subject to evolution or sudden changes to adapt to changing environments. The phenomenon is also called *concept drift* in data mining (Gama et al. 2014). Undetected and untreated, different variants of a single process over time can lead to overly complex process models and faulty analysis results.

Strategies: Divide and conquer. Partition event log based on underlying process.

Approaches: To partition the event log correctly, the concept drifts need to be localized first. One of the first attempts to detect concept drifts in business processes is Bose et al. (2014). By analyzing event logs and statistically comparing the relations between events in two adjacent sliding windows, significant differences can be detected and concept drifts can be detected. More advanced methods to detect process changes over time are described in Ostovar et al. (2017).

Event Log Characteristics

Besides the process characteristics, the circumstances of recording process execution data, as well as the extraction of data from systems, affect

the event log. Thus, the nature of event logs and their generation must be well understood to select and apply the appropriate event log cleaning techniques.

Different Processes. When extracting process data from information systems, the extracted event logs can contain events from different processes or different process variants.

Strategies and Approaches: See point *Variability* in Sect. 1.

Different Sources. Real business processes are often supported by multiple different IT systems. Hence, the number and heterogeneity of sources from which data is integrated in the event log can complicate the analysis (Song et al. 2016). Dealing with multiple sources can create temporal inconsistencies, semantic ambiguities, or also different levels of granularity of recording. Because each of these issues can also occur with only a single event source, they are listed as separate points.

Strategies: Awareness of potential issues. Apply tailored data cleaning solution.

Approaches: The unification of different sources is largely a manual and time-consuming task. This is an ongoing research challenge, and mostly partial solutions to subproblems exist. The analyst needs to pick the appropriate techniques to transform the different sources into a common format. Approaches exist in the domain of schema matching (Rahm and Bernstein 2001).

Time stamps. When event logs are integrated from different sources, inconsistencies can be caused by the clocks of different systems being out of sync. Untreated, these temporal issues may lead to incorrect ordering of events in a unified event log. Currently, this problem requires the analyst to investigate potential time differences within systems and apply a correction to restore synchronicity. If the systems are spread over different time zones, additional complexity in harmonizing the time stamps stems from different treatment of daylight savings time.

Strategies: Detect anomalies. Repair data.

Approaches: Correct time stamps based on rules. Inconsistencies can be detected by anomaly detection methods (e.g., using probabilistic models (Rogge-Solti and Kasneci 2014) or temporal constraints (Song et al. 2016))

Correlation. Typically, events of a single business case can be identified by a correlating attribute. For example, the “order id” of a customer order identifies the case, and all related events carry this information as attribute. In some cases, however, this information is missing, and it can be unclear how to correlate events in an event log.

Strategies: Find correlation anchors (the correlation anchor can change within a case). Repair data.

Approaches: In the simpler case, correlation is possible through attributes. An approach to identify correlation attributes is Nezhad et al. (2011). It can be used for correlating events belonging to the same cases. In the more complicated case, such attributes cannot be found. Even then, there are attempts to solve this problem (Bayomie et al. 2016; Pourmirza et al. 2017). Latter techniques work with dependencies between activities and try to find a mapping between cases and events that is consistent (Bayomie et al. 2016) or even optimal with respect to the temporal variations in the process (Pourmirza et al. 2017).

Completeness. Completeness issues can appear on different aggregation levels, meaning that entire cases, single events, or also event attributes can be missing from an event log.

Strategies: Impute missing data. Remove incomplete cases.

Approaches for missing cases: It can happen that entire cases are missing from event logs. The detection of missing cases is nontrivial, however. A chance to detect missing cases is by carefully analyzing idle times of resources for patterns of successive idleness. These “blank spots” in the schedules could indicate a missing case. Further indications might be found in cross-checking the event log records with other sources (e.g., the accounting books).

Approaches for missing events: If single events are missing from an event log, causal relations between events can be used to deduce which events are missing. For example, a case with a “leave room” event, but without a logically preceding “enter room” event, points at a missing record. Logic-based missing event completion techniques are used in Bertoli et al. (2013) and Francescomarino et al. (2015). Also process models can be used to fill in the blanks (de Leoni et al. 2015; Wang et al. 2016).

Approaches for missing attributes: The problem of missing attributes is especially common, when the event documentation is done by humans. Two approaches are common: the minimum change principle (Song et al. 2016; Mannhardt et al. 2016a) or the maximum likelihood principle (Rogge-Solti et al. 2013; Yakout et al. 2013).

Granularity. There is not always a 1:1 relationship between events in an event log and business activities that analysts are interested in. Events can be recorded in a higher frequency than required. Also the contrary can be the case. For example, temporal information can be cloaked or recorded on a daily granularity only.

Strategies: Group low-level events to higher level activities. Infer relations.

Approaches:

- (a) Dealing with the event granularity mismatch.

A bridge between the low-level events to the high-level business events needs to be built by establishing a mapping. Rule-based approaches were the first to address the problem of n:m relations between events and activities (Baier et al. 2014). If low-level events follow a certain structure, a pattern-based mapping is possible between low-level events and high-level events (Mannhardt et al. 2016b). Also works based on raw sensor data exist that convert location data into process instances by solving an optimization problem and taking potential

process knowledge into account (e.g., temporal, resource assignments, precedes relationships) (Senderovich et al. 2016).

- (b) Dealing with the attribute granularity mismatch. When time stamps are coarse grained such that the order of events cannot be inferred by looking at the time stamps, events need to be treated as *partially ordered*. One idea is to guess the order of events by exploiting ordered information from other cases within the event log (Lu et al. 2014).

Ambiguity. Event labels can be ambiguous, and the relation between business activity in a process model and event is unclear. For example, two events with the same label (or name) can indicate different activities in a process depending on their context.

Strategies: Relabel ambiguous terms.

Approaches: Semantic ambiguities are a highly complex problem, and only specific solutions exist, but techniques to relabel certain activities with the same label based on their context greatly facilitate the process mining and analytics part (e.g., Song et al. 2009; de San Pedro and Cortadella 2016).

Further Reading

In this handbook, a condensed view on event log cleaning methods is provided. Deeper insights can be retrieved from the text book on process mining (van der Aalst 2016, Chapter 4). Additionally, the *process mining manifesto* emphasizes the need for handling of event log quality issues to improve process analytics results. It defines a five-level maturity model for the assessment of event logs. The scale ranges from “★” lowest quality (e.g., paper-based medical records) to “★★★★★” highest quality (e.g., semantically annotated logs of BPM systems) (van der Aalst et al. 2011).

A collection of so-called event log *imperfection patterns* lists common problems that occur in real projects and also provides recipes

to overcome these problems (Suriadi et al. 2017). Another related work specifically addresses the preprocessing of event logs required for process mining (Bose et al. 2013). Last, the approach proposed in (de Leoni et al. 2016) can solve data cleaning problems on the basis of data mining tools by projecting event logs into data tables. This approach enables a large share of data manipulation and analytics tool set of the data mining field (Han et al. 2011) in the process mining context.

Conclusion

The quality of an event log is of utmost importance for accurate process analytics. Data cleaning needs to be done with care, and each performed step needs to be documented. Furthermore, access to the raw event data can help to evaluate the effect of the data cleansing steps and enables analysts to replace certain cleaning approaches. Automatically handling different quality issues is an active research field with more and more novel and improving approaches appearing recently.

Cross-References

- ▶ [Conformance Checking](#)
- ▶ [Data Quality and Data Cleansing of Semantic Data](#)
- ▶ [Data Cleaning](#)

References

- Baier T, Mendling J, Weske M (2014) Bridging abstraction layers in process mining. *Inf Syst* 46:123–139. <https://doi.org/10.1016/j.is.2014.04.004>
- Bayomie D, Awad A, Ezat E (2016) Correlating unlabeled events from cyclic business processes execution. In: Advanced information systems engineering – 28th international conference, CAiSE 2016, Ljubljana, 13–17 June 2016. Proceedings, pp 274–289. https://doi.org/10.1007/978-3-319-39696-5_17
- Bertoli P, Francescomarino CD, Dragoni M, Ghidini C (2013) Reasoning-based techniques for dealing with incomplete business process execution traces. In: AI*IA 2013: advances in artificial intelligence – XIIth international conference of the Italian association for artificial intelligence, Turin, 4–6 Dec 2013. Proceedings, pp 469–480. https://doi.org/10.1007/978-3-319-03524-6_40
- Bose JCJC, Mans RS, van der Aalst WMP (2013) Wanna improve process mining results? In: IEEE symposium on computational intelligence and data mining, CIDM 2013, Singapore, 16–19 Apr 2013, pp 127–134. <https://doi.org/10.1109/CIDM.2013.6597227>
- Bose RPJC, van der Aalst WMP, Zliobaite I, Pechenizkiy M (2014) Dealing with concept drifts in process mining. *IEEE Trans Neural Netw Learn Syst* 25(1):154–171. <https://doi.org/10.1109/TNNLS.2013.2278313>
- Conforti R, Rosa ML, ter Hofstede AHM (2017) Filtering out infrequent behavior from business process event logs. *IEEE Trans Knowl Data Eng* 29(2):300–314. <https://doi.org/10.1109/TKDE.2016.2614680>
- de Leoni M, Maggi FM, van der Aalst WMP (2015) An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf Syst* 47:258–277. <https://doi.org/10.1016/j.is.2013.12.005>
- de Leoni M, van der Aalst WMP, Dees M (2016) A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Inf Syst* 56:235–257. <https://doi.org/10.1016/j.is.2015.07.003>
- de Lima Bezerra F, Wainer J (2013) Algorithms for anomaly detection of traces in logs of process aware information systems. *Inf Syst* 38(1):33–44. <https://doi.org/10.1016/j.is.2012.04.004>
- de San Pedro J, Cortadella J (2016) Discovering duplicate tasks in transition systems for the simplification of process models. In: Business process management – 14th international conference, BPM 2016, Rio de Janeiro, 18–22 Sept 2016. Proceedings, pp 108–124. https://doi.org/10.1007/978-3-319-45348-4_7
- Diamantini C, Genga L, Potena D, van der Aalst WMP (2016) Building instance graphs for highly variable processes. *Expert Syst Appl* 59:101–118. <https://doi.org/10.1016/j.eswa.2016.04.021>
- Dumas M, Rosa ML, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer, <https://doi.org/10.1007/978-3-642-33143-5>
- Francescomarino CD, Ghidini C, Tessaris S, Sandoval IV (2015) Completing workflow traces using action languages. In: Advanced information systems engineering – 27th international conference, CAiSE 2015, Stockholm, 8–12 June 2015, Proceedings, pp 314–330. https://doi.org/10.1007/978-3-319-19069-3_20
- Gama J, Zliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv* 46(4):44:1–44:37. <http://doi.acm.org/10.1145/2523813>
- Greco G, Guzzo A, Pontieri L, Saccà D (2006) Discovering expressive process models by clustering log traces. *IEEE Trans Knowl Data Eng* 18(8):1010–1027. <https://doi.org/10.1109/TKDE.2006.123>

- Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann, <http://hanj.cs.illinois.edu/bk3/>
- International Organization for Standardization (2011) Software engineering – Software Product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE
- Leemans SJ (2017) Robust process mining with guarantees. Ph.D thesis, Eindhoven University of Technology. https://pure.tue.nl/ws/files/63890938/20170509_Leemans.pdf
- Lu X, Fahland D, van der Aalst WMP (2014) Conformance checking based on partially ordered event data. In: Business process management workshops – BPM 2014 international workshops, Eindhoven, 7–8 Sept 2014, Revised papers, pp 75–88. https://doi.org/10.1007/978-3-319-15895-2_7
- Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP (2016a) Balanced multi-perspective checking of process conformance. Computing 98(4):407–437. <https://doi.org/10.1007/s00607-015-0441-1>
- Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP, Toussaint PJ (2016b) From low-level events to activities – a pattern-based approach. In: Business process management – 14th international conference, BPM 2016, Rio de Janeiro, 18–22 Sept 2016. Proceedings, pp 125–141. https://doi.org/10.1007/978-3-319-45348-4_8
- Mans RS, Schonenberg H, Song M, van der Aalst WMP, Bakker PJM (2008) Application of process mining in healthcare – a case study in a Dutch hospital. In: Biomedical engineering systems and technologies, international joint conference, BIOSTEC 2008, Funchal, Madeira, 28–31 Jan 2008, Revised selected papers, pp 425–438. https://doi.org/10.1007/978-3-540-92219-3_32
- Nezhad HRM, Saint-Paul R, Casati F, Benatallah B (2011) Event correlation for process discovery from web service interaction logs. VLDB J 20(3):417–444. <https://doi.org/10.1007/s00778-010-0203-9>
- Ostovar A, Maaradji A, Rosa ML, ter Hofstede AHM (2017) Characterizing drift from event streams of business processes. In: Advanced information systems engineering – 29th international conference, CAiSE 2017, Essen, 12–16 Jun 2017, Proceedings, pp 210–228. https://doi.org/10.1007/978-3-319-59536-8_14
- Pourmirza S, Dijkman RM, Grefen P (2017) Correlation miner: mining business process models and event correlations without case identifiers. Int J Coop Inf Syst 26(2):1–32. <https://doi.org/10.1142/S0218843017420023>
- Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350. <https://doi.org/10.1007/s007780100057>
- Reichert M, Weber B (2012) Enabling flexibility in process-aware information systems – challenges, methods, technologies. Springer, <https://doi.org/10.1007/978-3-642-30409-5>
- Rogge-Solti A, Kasneci G (2014) Temporal anomaly detection in business processes. In: Business process management – 12th international conference, BPM 2014, Haifa, 7–11 Sept 2014. Proceedings, pp 234–249. https://doi.org/10.1007/978-3-319-10172-9_15
- Rogge-Solti A, Mans R, van der Aalst WMP, Weske M (2013) Improving documentation by repairing event logs. In: The practice of enterprise modeling – 6th IFIP WG 8.1 working conference, PoEM 2013, Riga, 6–7 Nov 2013, Proceedings, pp 129–144. https://doi.org/10.1007/978-3-642-41641-5_10
- Senderovich A, Rogge-Solti A, Gal A, Mendling J, Mandelbaum A (2016) The ROAD from sensor data to process instances via interaction mining. In: Advanced information systems engineering – 28th international conference, CAiSE 2016, Ljubljana, 13–17 Jun 2016. Proceedings, pp 257–273. https://doi.org/10.1007/978-3-319-39696-5_16
- Song JL, Luo TJ, Chen S, Liu W (2009) A clustering based method to solve duplicate tasks problem. J Grad School Chin Acad Sci 26(1):107–113
- Song S, Cao Y, Wang J (2016) Cleaning timestamps with temporal constraints. VLDB 9(10):708–719. <http://www.vldb.org/pvldb/vol9/p708-song.pdf>
- Suriadi S, Andrews R, ter Hofstede AHM, Wynn MT (2017) Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf Syst 64:132–150. <https://doi.org/10.1016/j.is.2016.07.011>
- van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer, <https://doi.org/10.1007/978-3-662-49851-4>
- van der Aalst WMP, Adriansyah A, de Medeiros AKA, Arcieri F, Baier T, Bickle T, Bose RPJC, van den Brand P, Brandtjen R, Buijs JCAM, Burattin A, Carmona J, Castellanos M, Claeis J, Cook J, Costantini N, Curbela F, Damiani E, de Leoni M, Delias P, van Dongen BF, Dumas M, Dustdar S, Fahland D, Ferreira DR, Gaaloul W, van Geffen F, Goel S, Günther CW, Guzzo A, Harmon P, ter Hofstede AHM, Hoogland J, Ingvaldsen JE, Kato K, Kuhn R, Kumar A, Rosa ML, Maggi FM, Malerba D, Mans RS, Manuel A, McCreesh M, Mello P, Mendling J, Montali M, Nezhad HRM, zur Muehlen M, Munoz-Gama J, Pontieri L, Ribeiro J, Rozinat A, Pérez HS, Pérez RS, Sepúlveda M, Sinur J, Soffer P, Song M, Sperduti A, Stilo G, Stoel C, Swenson KD, Talamo M, Tan W, Turner C, Vanthienen J, Varvaressos G, Verbeek E, Verdonk M, Vigo R, Wang J, Weber B, Weidlich M, Weijters T, Wen L, Westergaard M, Wynn MT (2011) Process mining manifesto. In: Business process management workshops – BPM 2011 international workshops, Clermont-Ferrand, 29 Aug 2011, Revised selected papers, part I, pp 169–194, https://doi.org/10.1007/978-3-642-28108-2_19
- Wang J, Song S, Zhu X, Lin X, Sun J (2016) Efficient recovery of missing events. IEEE Trans Knowl Data Eng 28(11):2943–2957. <https://doi.org/10.1109/TKDE.2016.2594785>
- Yakout M, Berti-Équille L, Elmagarmid AK (2013) Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In: Proceedings of the ACM SIGMOD international conference on

management of data, SIGMOD 2013, New York, 22–27 Jun 2013, pp 553–564. <http://doi.acm.org/10.1145/2463676.2463706>

Exploratory Data Analytics

► Visualization Techniques

Event Log Preprocessing

- Event Log Cleaning for Business Process Analytics

Event Log Repair

- Event Log Cleaning for Business Process Analytics

Event Pattern Recognition

- Pattern Recognition

Event Streams

- Definition of Data Streams

Eventual Consistency

- TARDiS: A Branch-and-Merge Approach to Weak Consistency

Exploratory Data Analysis

- Big Data Visualization Tools

Exploring Scope of Computational Intelligence in IoT Security Paradigm

E

Soumya Banerjee¹, Samia Bouzefrane², and Hanene Maupas³

¹Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, India

²Conservatoire National des Arts et Metiers, Paris, France

³IDEAMIA Colombes, Colombes, France

Definitions

As the expansion of nano-devices, smartphones, 5G, tiny sensors, and distributed networks evolve, the IoT is combining the “factual and virtual” anywhere and anytime. Subsequently, it is attracting the attention of both “maker and hacker.” However, interconnecting many “things” also means the possibility of interconnecting many diversified threats and attacks. For example, a malware virus can easily propagate through the IoT at an unprecedented rate. In the four design aspects of the smart IOT, there may be various threats and attacks; they are:

- (a) Data perception and collection: In this aspect, typical attacks include data leakage, sovereignty, breach, and authentication.
- (b) Data storage: The following attacks may occur – denial-of-service attacks (attacks on availability), access control attacks, integrity attacks, impersonation, modification of sensitive data, and so on.
- (c) Data processing: In this aspect, there may exist computational attacks that aim to generate wrong data processing results.

- (d) Data transmission: Possible attacks include channel attacks, session hijacks, routing attacks, flooding, and so on. Apart from attenuation, theft, loss, breach, and disaster, data can also be fabricated and modified by the compromised sensors.

Health-related and medical data is one of the sensitive application paradigms, where smart healthcare IoT deals with voluminous and meaningful data. As health data are highly critical for personal privacy and relevant regulations such as HIPAA (Lever et al. 2015) must be conformed to, the uploaded data need to be encrypted. The privacy of the communication link between smartphones and cloud servers can be protected by underlying media access control (MAC) protocols. A straightforward method is to encrypt uploaded data with off-the-shelf methods such as block ciphers, for example, AES or KASUMI (Ould-Yahia et al. 2017; Li et al. 2015). However, this may not be suitable and applicable in smartphones, because smartphones usually have energy constraints. Moreover, smartphones may be misused, lost, stolen, or hacked by attackers; thus privacy protection itself should be robust. Therefore, it is a critical challenge to design a lightweight and robust method to protect privacy.

As all the phases of data transmission of IoT involves substantial amount of threat and vulnerability, hence therefore trust and authentication in smart system have become crucial. Specifically, we envisage the smart health sector, where the medical data and health data is deployed through various involvement and stages influenced by human factors (e.g., doctors, patients, several medical agencies, diagnostic centers). The categories of known and unknown attacks improvise to develop more adaptive and self-organized system, which could foster into embedded smart applications. Conventional techniques like encryption and security protocol may not provide adequate security support. Typically, secure embedded systems are guided by the factors, e.g., small form factor, high performance, low energy consumption (Urien 2016), and robustness to attacks. The prediction, decision planning, and action can be

synchronized in these sections of health centric IoT application.

This proposal explores the possibilities and realization of computational intelligence techniques in terms of empowering trust, authentication, and residence mechanism. The desired objective is to configure the trusted and secure elements for smart IOT by using fuzzy logic, Bayesian attack graphs, intelligent learning, hybrid intelligent methods (e.g., combining fuzzy and neural network), and natural computing (like specifically through ant colony, bee colony optimization, swarm intelligence). The highlight of computational intelligence is to provide the security measures and action plan against unknown and uncertain threats and attacks considering the design constraints of secure elements (low energy consumption and high performance). The level of trust can be measured and data mined from different profile of users, contexts, and places of application deployment. However, depending on the design and adaptability of applications, even one or more intelligent techniques can be embedded in the design of single secure elements. The level of secure elements can also be ranked depending on the deployment intelligent techniques from lower to higher security levels. Till this novel article is formulated, there was no documented chronological survey available of such prototypes available.

State-of-the-Art & Contemporary Applications

As mentioned, there are many existing research and industrial applications to counter different security attacks for connected objects. However, most of them are dependent either on conventional cryptography or on secured protocol design (Zhao 2013; Bao and Chen 2012; Lize et al. 2014; Yan et al. 2014). However, there are few approaches, where fuzzy logic has been used for trust management and access control (Mahalle et al. 2013). Still, commercialization of those intelligent security measures is not directly implemented with IoT and smart applications (WhitePaper 2017). Interestingly, there is good

zero-day safety matrix proposed to rank the degree of vulnerabilities (Wang et al. 2014). This mapping also can be made predictable. The present proposal emphasizes on the aspect of prediction and configuring a decision support system. In addition, it will be worthy to scan the detailed applications or products that exist with respect to relevant industrial perspective. The next section elaborates this issue.

Subsequently, there is substantial scope to configure and customize the existing suit of Smart IoT protocols. Considering the low energy consumption as prime criteria, out of the followings, some of them can be focused for intelligent algorithm and customization.

- CoAP: Constrained Application Protocol.
- MQTT: Message Queue Telemetry Transport.
- XMPP: Extensible Messaging and Presence Protocol.
- RESTFUL Services: Representational State Transfer.
- AMQP: Advanced Message Queuing Protocol
- Websockets

Computational Intelligence: A Road Map for Security in IoT and Connected Systems

The road map of using computation intelligence in this proposal could be as follows:

Fuzzy Logic-Based Trust and Authentication System

While developing a trust and authentication matrix using *Fuzzy Logic* (where uncertainty about the trust focused on the user can be measured and discrete value of trust can be evaluated. Embedded software for secure elements can be developed by embedded programming environment), the embedded fuzzy program with fuzzy operators can work on user behavior and profile data and can aggregate a discrete decision from fuzzy referral index, known as membership function. The secure elements realized as secure microcontrollers with low power consumption(<200 mW) and a

crypto processor that accelerates cryptographic processing. However fuzzy logic-driven secure elements can be positioned with Datagram Transport Layer Security (DTLS) or Transport Layer Security (TLS) stacks. The trust application can be scripted in *Python Data gram Transport Layer Security* (<https://pypi.python.org/pypi/Dtls>) and as a whole is well supported by SSL. However, for secure elements, JAVACARD Syntax is popularly used, which is a subset of JAVA language (Javaid et al. 2016).

E

Concept of Attack Graph for Interconnected Systems

Generation of attack graph is recently used to assess the risk of interconnected networks (Lever et al. 2015), applying *Bayesian attack graphs* at real time on secure elements to calculate the probability that an attacker can reach each state (condition) in the graph. Instead of attempting to rank unknown vulnerabilities, the proposed BAG metric counts how many such vulnerabilities would be required for compromising network assets (Wang et al. 2014); a larger count implies more security since the likelihood of having more unknown vulnerabilities available, applicable, and exploitable all at the same time will be significantly lower. Categorically, the IoT malware families, e.g., BASHLITE and Mirai, can be analyzed to determine common properties which can be used for detection through probability values of BAG. Even an attribute data base can be prepared offline for other IoT malware families such as Hajime, NyaDrop, and LuaBot, this will assist to classify the types of malware attacks on line. The malware mentioned is also known as botnet of IoT: BAG algorithm can be developed and emulated through observe-orient-decide-act (OODA) to match the requirements of smart healthcare applications. This will be a decision dashboard, and it assists IoT management. Finally, a security risk measure matrix can be evaluated, and based on the recommendation, OODA can be triggered. High-level programming description will be sufficient.

Hybrid Model and Intelligent Learning

In this paradigm of computational intelligence, more than one intelligent model can be combined to ensure the detection of malware, attack-prone component, and intrusion. For example, artificial fish swarm algorithm (Lin et al. 2014) can be implemented with analyzing various features of possible intrusion. This could be distance, vision, neighbor, center, crowded degree, and maximum iterations. Here, considering (support vectors can be used for classifying new data) F_i represents fish i and C_i represents the center of F_i , then the embedded applications for configuring secure elements are:

- Initialization
- Fitness Evaluation
- Movement dynamics of fish swarms

At first, a feature subset is randomly assigned to N fishes. All parameters including vision, maximal crowded degree, and maximal trial number are defined. For example, if eight fishes were initiated, each fish has its own feature, and the circle represents the vision of fish i . The next step is to evaluate the classification value as a fitness value of the feature subset of each fish.

Starting with the first fish, follow step is implemented. If follow is successful, optimal value of fitness of fish demonstrates highly secured indicator; for example, in Fig. 1, the fitness value of fish i is 55; by contrast, the best fitness neighbor exhibits a value of 80. Thus, the best fitness neighbor demonstrates a superior fitness value, indicating that a superior fish is located in the vision of fish i . Therefore, the follow step is successful, and fish i moves to the location of best fitness neighbor and can be declared as secure zone in real time (Refer Fig. 1).

Trusted and low power objects ideally be based on Linux operating system, for example, Raspberry Pi board with DEBIAN distribution. Here, through middleware, the proposed algorithm could be developed (Beal et al. 2015; Viroli et al. 2016) by a recent emerging technique known as “Aggregate Programming” (Beal et al. 2015). Potential attack can be detected and

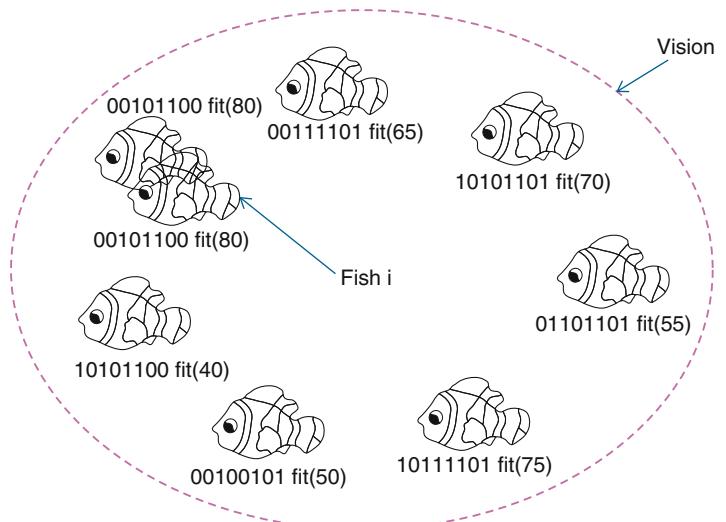
warnings disseminated robustly with just a few lines of Protelis code dynamically deployed and executed on individual devices by a middleware application. This short program is resilient and adaptive, enabling it to effectively estimate for identifying the prone to attacks (could be partitioned in several fuzzy values like low, moderately low, depending medium, very high, etc.) and distribute warnings while executing on numerous mobile devices. Mobile Python could be ideally chosen for embedding the application for secure elements.

Natural Computing for Resilience and Self-Organized Mechanism

The vertical applications of natural computing are driven by social insects like ants, bees, or swarms. Modeling of ant colony system (ACS) under random undirected graph can also define IoT configuration and empirical measurement of trust and security. Measured values are evaluated from topological mapping of pheromone communication of ants across the graph of interconnected sensors (Ould-Yahia et al. 2017). In the dynamic contexts of sensors (in case of body wearable sensors), it will be more appropriate to quantify contextual or referential intelligence according to the locations of the sensors. The degree of security or trust can vary with respect to the sensors’ locations.

Computational intelligence can be plugged into smart health security. Smart health of interconnected devices often needs intelligent control policies (Ayala et al. 2012) to manage different contexts of IoT sensors to sense. Examples of such capabilities are control policies for managing a self-controlled swarm of IoT devices (Viroli et al. 2016) (hierarchical vs. collaborative control strategies) or context-aware control of devices and adaptation of their behavior according to the environment. The mobile wireless body sensor and participatory sensing. Mobile WBSN comprises multiple sensor nodes that are implanted (or attached) into (or on) a human body to monitor health or EEG physiological indicators, such as electrocardiogram (ECG electroencephalography), glucose, toxins, blood pressure,

Exploring Scope of Computational Intelligence in IoT Security Paradigm, Fig. 1
Dynamic movement of fish for connected systems.
 (Scheme Courtesy Lin et al. 2014)



E

and so on. Data usually need to be uploaded to a central database (e.g., a cloud computing server) instantly so that doctors or nurses can remotely access them for real-time diagnosis and emergency response. As most persons possess a smartphone and customized applications can be installed on it, it is convenient and economical to use a smartphone as a gateway between WBSN and cloud servers. Participatory sensing also follows the suitable authentication scheme, and here the self-organized swarms can be programmed to deploy for access control of the application. Conventionally, one-time mask (OTM) scheme and one-time permutation (OTP) scheme both are reasonably efficient from energy consumption of view of IoT system, to protect the data of WBSN. However, as an additional security cover, swarm agents are used to anticipate the sensing scheme. Based on the design of swarm control, these encryption schemes can be functional.

Currently, there are agent technologies that can be embedded in devices of the IoT, such as Jade-Leap and Agent Factory Micro Edition (Muldoon et al. 2006). The embedded development may consider multiple agent technologies, like Jade (Muldoon et al. 2006), Jadex, Jade-Leap, and Mobile Agent Platform (Bellifemine et al. 2001). In this case, the agents are the managers of the sensors and make these data available to the IoT. Thus they can be adapted to

new devices (i.e., Symbian, Android, Sun SPOT, Libelium) and network technologies (i.e., WiFi, Bluetooth, ZigBee, XBee, NFC) and extended with additional features, such as goal orientation (Aiello et al. 2011) and self-management (Ayala et al. 2013).

Conclusion and Scope of Future Research

The paradigm of interconnected devices, sensors, and IoT improvises manifold applications with the requirement of substantial security cover to facilitate the big data-driven interactions. Static measures and conventional form of security, e.g., cryptography, may not be effective. Hence, different dimensions of computationally intelligent systems can be deployed in IoT environment, and importantly, the approaches are well supported with embedded system components for IoT. Approximation measures (fuzzy logic-based system and attack graphs), automated learning, and social insect-driven approaches are already been implemented, and thus new generation of IoT can be more dynamically secured and protected from different anticipated security breaches. At present, more machine learning-based methodologies like deep learning (Javaid et al. 2016;

Li et al. 2015) are planned to be incorporated with malicious code detection in IoT applications.

References

- Aiello F, Fortino G, Gravina R, Guerrieri A (2011) A java-based agent platform for programming wireless sensor networks. *Comput J* 54(3):439–454
- Ayala I, Pinilla MA, Fuentes L (2012) Exploiting dynamic weaving for self-managed agents in the IOT. In: German conference on multiagent system technologies. Springer, pp 5–14
- Ayala I, Amor M, Fuentes L (2013) Self-configuring agents for ambient assisted living applications. *Pers ubiquit comput* 17(6):1159–1169
- Bao F, Chen IR (2012) Dynamic trust management for internet of things applications. In: Proceedings of the 2012 international workshop on Self-aware internet of things. ACM, pp 1–6
- Beal J, Pianini D, Viroli M (2015) Aggregate programming for the internet of things. *Computer* 48(9):22–30
- Bellifemine F, Poggi A, Rimassa G (2001) Developing multi-agent systems with a fipa-compliant agent framework. *Softw Pract Exp* 31(2):103–128
- Javid A, Niyaz Q, Sun W, Alam M (2016) A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (formerly BIONETICS), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp 21–26
- Lever KE, Kifayat K, Merabti M (2015) Identifying inter-dependencies using attack graph generation methods. In: 2015 11th international conference on innovations in information technology (IIT). IEEE, pp 80–85
- Li Y, Ma R, Jiao R (2015) A hybrid malicious code detection method based on deep learning. *Methods* 9(5):205–216
- Lin KC, Chen SY, Hung JC (2014) Botnet detection using support vector machines with artificial fish swarm algorithm. *J Appl Math* 2014:1–9
- Lize G, Jingpei W, Bin S (2014) Trust management mechanism for internet of things. *China Commun* 11(2):148–156
- Mahalle PN, Thakre PA, Prasad NR, Prasad R (2013) A fuzzy approach to trust based access control in internet of things. In: 2013 3rd international conference on wireless communications, vehicular technology, information theory and aerospace & electronic systems (VITAE). IEEE, pp 1–5
- Muldoon C, O'Hare GM, Collier R, O'Grady MJ (2006) Agent factory micro edition: a framework for ambient applications. In: International conference on computational science. Springer, pp 727–734
- Ould-Yahia Y, Banerjee S, Bouzefrane S, Boucheneb H (2017) Exploring formal strategy framework for the security in iot towards e-health context using computational intelligence. In: Internet of things and big data technologies for next generation healthcare. Springer, pp 63–90
- Urien P (2016) Three innovative directions based on secure elements for trusted and secured iot platforms. In: 2016 8th IFIP international conference on new technologies, mobility and security (NTMS). IEEE, pp 1–2
- Viroli M, Audrito G, Damiani F, Pianini D, Beal J (2016) A higher-order calculus of computational fields. arXiv preprint. arXiv:161008116
- Wang L, Jajodia S, Singhal A, Cheng P, Noel S (2014) k-zero day safety: a network security metric for measuring the risk of unknown vulnerabilities. *IEEE Trans Dependable Secure Comput* 11(1):30–44
- WhitePaper (2017) Iot 2020: smart and secure. International Electrotechnical Commission, Geneva
- Yan Z, Zhang P, Vasilakos AV (2014) A survey on trust management for internet of things. *J Netw Comput Appl* 42:120–134
- Zhao YL (2013) Research on data security technology in internet of things. *Appl Mech Mater Trans Tech Publ* 433:1752–1755

Extensible Record Store

► NoSQL Database Systems

Extract-Transform-Load

► ETL

Feature Learning from Social Graphs

Vineeth Rakesh¹, Lei Tang², and Huan Liu¹

¹Data Mining and Machine Learning Lab,
School of Computing, Informatics, and Decision
Systems Engineering, Arizona State University,
Tempe, AZ, USA

²Clari Inc., Sunnyvale, CA, USA

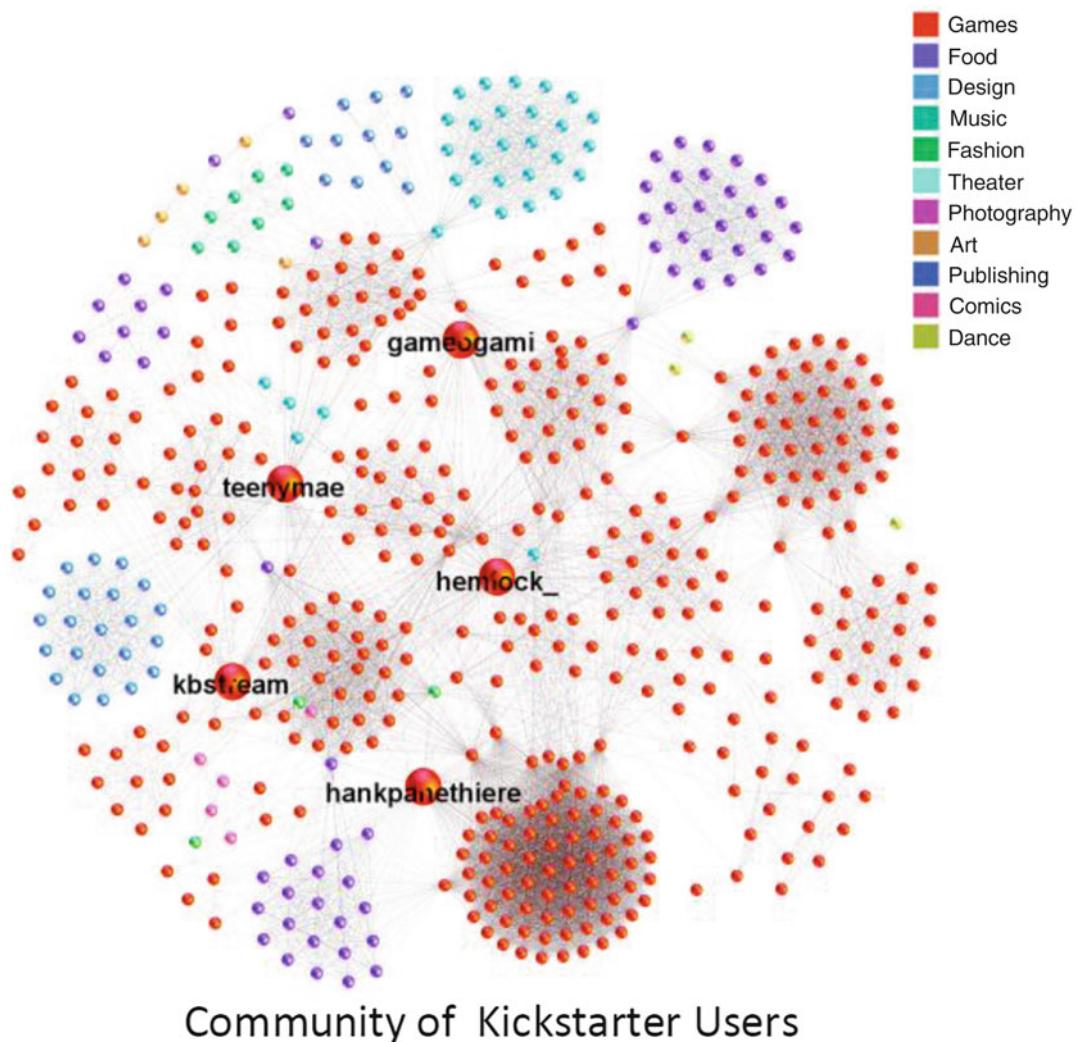
Overview

The rapid evolution of social network platforms such as Twitter, Facebook, and Instagram has resulted in heterogeneous networks with complex social interactions. Despite providing a rich source of information, the dimensionality of features in these networks poses several challenges to machine learning tasks such as personalization, prediction, and recommendation. Therefore, it is important to ask the question “how to capture such complex interactions between users in simplified dimensions?”. To answer this question, this entry explores network representation learning (NRL), where the objective is to model the complex high-dimensional interactions between nodes in a reduced feature space while simultaneously capturing the neighborhood similarity and community membership.

Information in the modern world flows in the form of graphs such as social networks, biological networks, and World Wide Web. These

graphical structures reveal many intriguing characteristics about the individuals and the structure of their communities. Figure 1 shows an example of a network that is composed of Twitter users who tweet about crowdfunding projects from Kickstarter (www.kickstarter.com). The colors in this graph indicate different communities that pertain to the category of projects like games, technology, art, etc. Communities are also known as *clusters* or *groups* (Jones et al. 2015), and they reveal several interesting properties of social networks such as homophily, cohesiveness, and structural holes. Consequently, the research on learning graph representations (or embedding) has gained significant attention in recent years where the objective is to represent the vertices of the graph in a low-dimensional vector space which embeds structural, semantic, and relational information.

Learning graph representations possesses two main challenges, namely, *heterogeneity* and *scalability*. Modern social networks are highly heterogeneous where interaction between the users cannot be measured using simple features (Tang et al. 2008). For instance, in Twitter, a user could interact with another user because both users follow the same celebrity or political figure (i.e., entity-centric interaction); in YouTube, the interaction between users might be due to their mutual interest in a video (i.e., content-centric interaction); and in location-based social networks such as Foursquare, interaction could be due to geo-location proximity between the users (i.e., location-centric interaction). Therefore, it is



Feature Learning from Social Graphs, Fig. 1 A Twitter-based community formed by the backers of Kickstarter (Rakesh et al. 2015). The colors represent

topical categories of communities, and the names denote the top users of the community

essential to develop models that can embed different layers of information in a condensed feature space. The second challenge is the scalability of models. Although various methods of graph embeddings have been proposed in the field of machine learning (Belkin and Niyogi 2002; Tenenbaum et al. 2000), these methods are suitable only for small networks with few thousand nodes and edges. Modern social networks on the other hand typically contain millions of nodes and billions of edges. For example, as of 2017, the popular social networking website Facebook has over 330

million users, while the microblogging website Twitter has over 2.7 billion users (www.statista.com). Therefore, it is extremely important to develop algorithms that scale to networks of this size.

To overcome the aforementioned challenges, researchers have proposed different algorithms that are highly effective in representing very large networks in a condensed feature space (Newman 2006b; Tang and Liu 2009; Perozzi et al. 2014; Grover and Leskovec 2016; Tang et al. 2015). This entry focuses on explaining three

most popular models, namely, SocDim (Tang and Liu 2009), DeepWalk (Perozzi et al. 2014), and node2vec (Grover and Leskovec 2016), that have become the state-of-the-art techniques in the field of network representation learning (NRL). These algorithms have proven to be highly scalable in capturing the latent representations from large graphs while simultaneously encoding the social similarity and community membership.

Definitions

The graph notation: A graph is denoted by $G = (V, E, F, Y)$, where $v \in V$ are the nodes (or actors), E are the edges of the network, $F \in \mathbb{R}^{|V| \times f}$ are the attributes, and f is the size of the feature space for each attribute and $Y \in \mathbb{R}^{|V| \times y}$ are the labels and y is the size of each label vector.

Random walks: Originally known as *Brownian motion* in space, the random walk is named after the botanist Robert Brown (www.wikipedia.org/wiki/Brownian_motion) who in 1826 peered at a drop of water using a microscope and observed tiny particles (such as pollen grains and other impurities) in it performing strange random-looking movements. A random walk on a undirected graph $G = (V, E)$ can be perceived as a stochastic process that starts from a given vertex and then selects one of its neighbors uniformly at random to visit. The transition probability from vertex v_i to v_j is defined as follows:

$$p(v_i, v_j) = \begin{cases} \frac{1}{Odeg(v_i)}, & v_j \text{ neighbor of } v_i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $Odeg(v_i)$ signifies the out-degree of node v_i . A random walk rooted at node v_i is denoted by W_{v_i} . Random walks are popular techniques for detecting the community and social structure of a network (Zafarani et al. 2014). In the subsequent sections, this entry will demonstrate how NRL algorithms utilize this technique to obtain representative samples of a large graph.

Key Research Findings

The Social Dimension Model

One of the earlier works on NRL was proposed by Tang and Liu (2009) to extract the latent dimensions (or features) that are indicative of the homophily in the network. To achieve this, the authors propose a model called SocDim that performs two separate functions: (a) extract latent social dimensions based on network connectivity and (b) construct a discriminative classifier that uses the latent features for predicting the node labels. Since the main focus of this entry is on network representation learning, the classifier part is excluded from the description. When developing algorithms for learning graph representations, it is important to make sure that the latent features capture the following properties:

- **Informative:** The social dimensions should be indicative of affiliations between actors.
- **Plural:** The same actor can get involved in multiple affiliations, thus appearing in different social dimensions.
- **Continuous:** The actors might have different degree of associations to one affiliation. Hence, a continuous value rather than a discrete $\{0, 1\}$ is more favorable.

Algorithm 1: SocDim(A,g,m)

```

1 Input: interaction matrix A, node degree g , number
   of edges m
2 Output: latent feature matrix  $\phi \in \mathbb{R}^{|V| \times d}$ 
3  $B = \text{GetModularityMat}(A, g, m)$ 
4  $\mathcal{E} = \text{GetTopEigenVec}(B)$ 
5 for each  $v \in V$  do
6   |  $\phi(v) = \text{GetLatentSocDim}(\mathcal{E}, v)$ 
7 end
```

Essentially, clustering techniques such as graph partitioning (Karypis and Kumar 1998), spectral clustering (Von Luxburg 2007), and block models (Nowicki and Snijders 2001) are the classic strategies for learning the latent representations of nodes in the network. However, these algorithms tend to ignore the

scale-free property (Chakrabarti and Faloutsos 2006) of large-scale social networks. To overcome this problem, SocDim uses the popular graph clustering algorithm called the Modularity (Newman 2006b) that explicitly takes degree distribution into consideration. Basically, modularity measures how far the interaction deviates from a uniform random graph with the same degree distribution. The objective is to divide a matrix A of v vertices and m edges into c nonoverlapping communities. For two nodes v_i and v_j with degree g_i and g_j , respectively, the modularity score is defined as follows:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{g_i g_j}{2m} \right] \delta(s_i, s_j) \quad (2)$$

where s_i and s_j denote the community membership of nodes i and j , respectively, and $\delta(s_i, s_j) = 1$ if $s_i = s_j$ and 0 otherwise. The term $\frac{g_i g_j}{2m}$ indicates the expected number of edges between these two nodes in a uniform random graph model. A positive Q value indicates that a cluster is able to capture the community structure, while a negative value indicates that the vertices are split into bad clusters. Therefore, the objective is to find a community structure such that Q is maximized. Maximizing the modularity over hard clustering is proven to be NP hard (Brandes et al. 2006); consequently, a soft clustering method is adopted. It is important to note that the adoption of soft clustering not only solves the problem efficiently but also generalizes well to the real-world scenario. That is, in real-world networks, users are not confined to one single community; instead, they have affiliations with different communities. The soft clustering is achieved by relaxing the notion of the community membership. Let $\mathbf{g} \in \mathbb{Z}_+^n$ be the degree of each node, and $S \in \{0, 1\}^{n \times k}$ be a community indicator matrix defined as follows:

$$S_{ij} = \begin{cases} 1 & i \text{ belongs to community } j \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The modularity matrix is defined as

$$B = A - \frac{\mathbf{g}\mathbf{g}^T}{2m} \quad (4)$$

using the above equation, the modularity can be reformulated as follows:

$$Q = \frac{1}{2m} \text{Tr}(S^T B S) \quad (5)$$

by relaxing S to be continuous, it can be shown that the optimal S is the top-k eigenvectors of the modularity matrix (Newman 2006a). Finally, there is one last bottleneck in solving the above equation, i.e., the modularity matrix B is dense and cannot be computed and held in memory when the number of nodes are large. To circumvent this issue, Lanczos method is used to calculate the top eigenvectors since it relies only on basic operation of matrix-vector multiplication. For further details on this method, readers are suggested to refer to Tang and Liu (2009). Algorithm 1 describes the three main steps of the SocDim model (lines 3–7), where line 3 uses Eq. 4 to obtain matrix B . The latent dimensions of nodes can be extracted using the top eigenvectors of the modularity matrix B .

The DeepWalk Model

The DeepWalk (Perozzi et al. 2014) is a popular model for network representation learning that excels in mapping large and complex graphs to a low-dimensional feature space. The algorithm begins by obtaining samples from a large graph using short random walks and then applies a neural language model over the samples to learn their latent representations. These latent representations are learned in such a way that the following characteristics are satisfied:

- **Evolution of networks:** The model should capture the continuously evolving nature of social networks.
- **Community aware:** The distance between latent dimensions should represent the social similarity between the members of the network.

- **Community membership:** The latent representations should model partial community membership in a continuous space.

One of the most interesting aspects of DeepWalk is the utilization of the *SkipGram* model (Mikolov et al. 2013) from the natural language processing (NLP) domain to capture the graph representations. The SkipGram tries to maximize the probability of observing a word w_n from a given sequence of words $\{w_0, w_1, \dots, w_{n-1}\}$. In other words, the algorithm tries to maximize the co-occurrence probability among the words that appear within a window w . A direct analogy can be established between SkipGram and NNL; i.e., given a node v_i , the objective is to maximize the likelihood of co-occurrence $p(v_i|(v_1, v_2, \dots, v_{i-1}))$. However, contrary to the SkipGram model, the primary objective of DeepWalk is not to estimate the likelihood of node co-occurrences but to capture the complex relationships between nodes in a low-dimensional feature space. To achieve this, DeepWalk introduces a subtle modification to the likelihood function. First, every node is represented by a real valued latent vector with a mapping function $\phi : v \in V \rightarrow \mathbb{R}^{|V| \times d}$, where d is the size of the latent dimension. Second, the objective is viewed as a problem of *predicting the whole context using a single missing word*. Therefore, the modified likelihood is written as follows:

$$\min_{\phi} \log Pr(\{v_{i-w}, \dots, v_{i-1}, v_{i+1}, \dots, v_{i+w}\} | \phi(v_i)) \quad (6)$$

To achieve this objective, DeepWalk uses a combination of truncated random walks and a neural language model shown in Algorithm 2. There are two main steps to this process. First, given a graph $G(V, E)$, at each vertex v_i , the algorithm begins by exploring its neighbors using a random walk of length ℓ (i.e., lines 5–8), and this procedure is repeated K times (the outer loop from lines 4–9). After generating a random walk $|W_{v_i}|$ of size ℓ , the algorithm then

Algorithm 2: DeepWalk(G, w, d, k, ℓ)

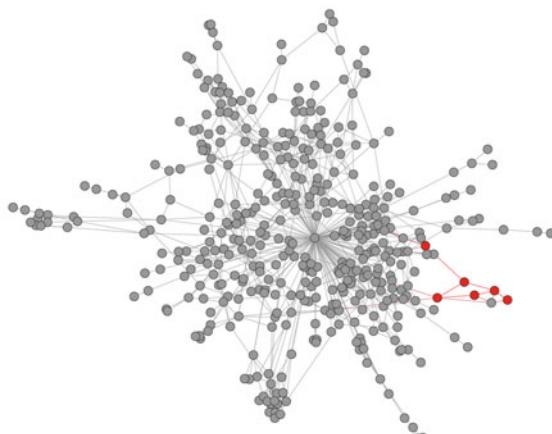
```

1 Input: graph  $G(V, E)$ , window size  $w$ , embedding size  $d$ 
2 # walks per node  $K$ , walk length  $\ell$ 
3 Output: latent feature matrix  $\phi \in \mathbb{R}^{|V| \times d}$ 
4 for  $i=0$  to  $K$  do
5   for each  $v_i \in V$  do
6      $\mathcal{W}_{v_i} = \text{RandomWalk}(G, v_i, \ell)$ 
7      $\text{SkipGram}(\phi, \mathcal{W}_{v_i}, w)$ 
8   end
9 end
10 SkipGram ( $\phi, \mathcal{W}_{v_i}, w$ )
11   for each  $v_j \in W_{v_i}$  do
12     for each  $\mu_k \in W_{v_i}[j-w:j+w]$  do
13        $J(\phi) = -\log Pr(\mu_k | \phi(v_j))$ 
14        $\phi = \phi - \alpha * \frac{\partial J}{\partial \phi}$ 
15     end
16   end
17 end

```

F

calls the *SkipGram* method in line 7. In the second step, the SkipGram procedure (line 10 in Algorithm 2) takes the random walk sample W_{v_i} as input, and for each node v_j , it iterates over all possible collocations of window length w in lines 11–12. These two steps are visually represented in Fig. 2a, b. Now, given the latent representation $\phi(v_j)$, the objective is to maximize the probability of obtaining a node μ_k from the neighborhood of v_j (line 13). However, this posterior is intractable to compute due to the marginalizing factor. To overcome this problem, a hierarchical softmax approximation technique is introduced. It works by constraining u_k to the number of nodes in the path from v_j to the leaves of the binary tree. This reframes the optimization problem into the task of maximizing the probability of a specific path in the tree; thereby, reducing the complexity of calculating the posterior from $O(|V|)$ to $O(\log|V|)$. This step is depicted in Fig. 2c. It should be noted that although the objectives of SocDim and DeepWalk are similar, they take very different approaches to learning the graph representations. The former directly optimizes for the modularity metric, while the latter first extracts random walk samples and then applies SkipGram to extract the latent dimensions.



(a) Random walk generation.

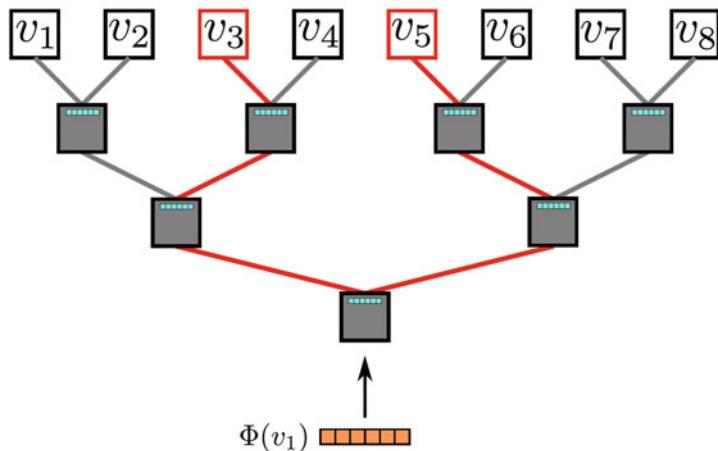
$$\mathcal{W}_{v_4} = \begin{matrix} 4 \\ 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{matrix}$$

$$u_k \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} v_j \longrightarrow \Phi(v_j)$$

Φ

The diagram illustrates the representation mapping step. It shows a vector u_k (represented by a column of numbers) being multiplied by a matrix v_j (represented by a row of numbers) to produce a latent feature vector $\Phi(v_j)$, which is represented as a grid of size $d \times j$.

(b) Representation mapping.



(c) Hierarchical Softmax.

Feature Learning from Social Graphs, Fig. 2 Shows the three main steps of DeepWalk algorithm where (a) denotes a random walk sample \mathcal{W}_{v_4} rooted at node v_4 , (b) shows the neighborhood window of node v_1 and the latent feature mapping $\phi(v_1)$ and (c) the hierarchical

softmax approximation technique which reduces the computational complexity of the posterior $Pr(v_j | \phi(v_1))$ by restricting the co-occurrence probability of v_1 to just v_3 and v_5 (Image Courtesy of Perozzi et al. 2014 and Perozzi 2016)

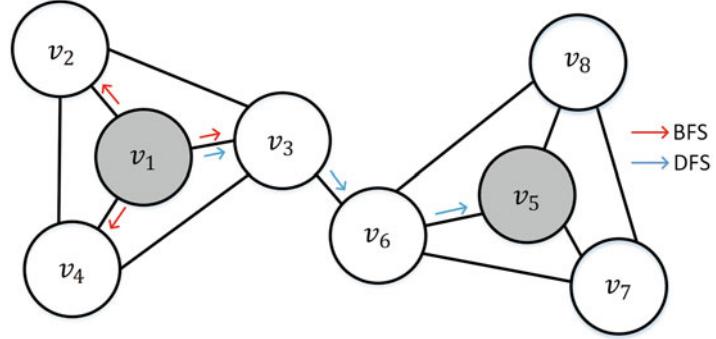
The node2vec Model

Similar to DeepWalk, the node2vec (Grover and Leskovec 2016) is another recent technique that learns representations of network by maximizing the likelihood of co-occurrence between a node and its neighborhood. An important drawback of DeepWalk is its sampling technique, which

is insensitive to the network connectivity patterns. To be more specific, the random walk method adopted by this model fails to capture community-specific properties such as *homophily* and *structural equivalence*. For example, Fig. 3 shows that node v_1 forms a tightly knit community with v_2, v_3 and v_4 (i.e., homophily) and node

Feature Learning from Social Graphs, Fig. 3

Example of BFS and DFS exploration strategies on a network exhibiting homophily and structural equivalence



F

v_5 forms a community with v_6 , v_7 , and v_8 . On the other hand, although v_1 and v_5 are from two distinct communities, they share the same structural equivalency by acting as hub nodes. Therefore, to capture such community-specific properties, the node2vec algorithm *defines a biased form of random walk* that aims to maximize the likelihood of preserving network neighborhood of nodes.

Given a node $v \in V$ and its neighbors $N_S(v) \subset V$ generated using a sampling strategy S , the nod2vec model aims to optimize the following function:

$$\max_{\phi} \sum \log Pr(N_S(v)|\phi(v)) \quad (7)$$

The above equation is very similar to the objective of the DeepWalk model (Eq. 6). Nonetheless, contrary to DeepWalk, which models the neighborhood $N_S(v)$ using a sliding window over consecutive nodes, the node2vec does not restrict the samples to the immediate neighbors of v . Instead, it allows to extract many different structures depending on the sampling strategy S .

Flexible biased random walk: As stated earlier, the aim of node2vec algorithm is to create sampling techniques that can capture both the structural equivalence and homophily in graphs. This is achieved using a combination of breadth-first search (BFS) and depth-first search (DFS) graph exploration strategies. BFS captures the structural equivalence by sampling the immediate neighbors of the given source node, while DFS captures the homophily between the nodes by sequentially sampling nodes at increasing distances from the source node. For example, in Fig. 3, for a

source node v_1 and neighborhood size $w = 3$, the BFS samples nodes v_2 , v_3 , v_4 and DFS samples v_3 , v_6 , v_5 . Furthermore, since BFS samples a very small section of a neighborhood, the nodes tend to repeat themselves in multiple neighborhood sets; consequently, this reduces the variance in characterizing the distribution of 1-hop nodes with respect to the source nodes. Contrary to this technique, the DFS samples neighborhood on a macro-level by exploring large parts of the network that results in capturing the homophily between the nodes.

Inspired by the above graph exploration strategies, the node2vec proposes a second-order random walk with a transition probability π that facilitates *smooth transitions between BFS and DFS*. The transition probability between nodes v_1 and v_2 is defined as follows:

$$\pi(v_1, v_2) = \begin{cases} \frac{1}{p} & \text{if } d_{v_1, v_2} = 0 \\ 1 & \text{if } d_{v_1, v_2} = 1 \\ \frac{1}{q} & \text{if } d_{v_1, v_2} = 2 \end{cases} \quad (8)$$

where d_{v_1, v_2} denotes the shortest path between v_1 and v_2 . Parameters p and q control how fast the walk explores and leaves the neighborhood of the starting node; in other words, they allow the search procedure to interpolate between BFS and DFS. The working of node2vec is illustrated in Algorithm 3. Similar to DeepWalk, the algorithm begins exploring its neighbors using a random walk of length ℓ , and this procedure is repeated K times (i.e., lines 7–11). The novelty of node2vec is described by the function *node2vecWalk* that

Algorithm 3: node2vec(G, d, K, ℓ, p, q)

```

1 Input: graph  $G(V, E)$ , latent dimension  $d$ , # walks
   per node  $K$ , context window  $w$ 
2 walk length  $\ell$ , return parameter  $p$ , In-out parameter
    $q$ 
3 Output: latent feature matrix  $\phi \in \mathbb{R}^{|V| \times d}$ 
4  $\pi = \text{GetTransitionProb}(G, p, q)$ 
5  $\text{walks} = []$ 
6  $G' = (V, E, \pi)$ 
7 for  $i=0$  to  $K$  do
8   for each  $u \in V$  do
9      $\text{walk} = \text{node2vecWalk}(G', u, \ell)$ 
10    Append  $\text{walk}$  to  $\text{walks}$ 
11  end
12   $\phi = SGD(w, \text{walks}, d)$ 
13 end
14  $\text{node2vecWalk}(G', u, \ell)$ 
15    $\text{walk} = [u]$ 
16   for each  $l \in \ell$  do
17      $cur = \text{walk}[-1]$ 
18      $V_{cur} = \text{GetNeighbors}(cur, G')$ 
19      $\text{sample} = \text{FlexRanWlk}(V_{cur}, \pi)$ 
20     Append  $\text{sample}$  to  $\text{walk}$ 
21   end
22   return  $\text{walk}$ 
23 end

```

Feature Learning from Social Graphs, Table 1
 Important differences between SocDim, DeepWalk, and node2vec models. Both DeepWalk and node2vec are highly scalable, but the latter is more sensitive to parameter changes

Model	Method	P-Sensitivity	Scalability
SocDim	Eign + Modularity	d	Limited
DeepWalk	Rwalk + SkipGrm	ℓ, d	High
node2vec	Rwalk* + SGD	ℓ, d p, q	High

performs the biased second-order random walk in line 19. After sampling several walks, the latent representation of nodes is learned using stochastic gradient descent (SGD) in line 12.

Table 1 delineates the key differences between the three models that were discussed in this entry. Here, $Rwalk$ and $Rwalk^*$ indicate the random walk and the modified random walk that is used by DeepWalk and node2vec, respectively. To conclude, the algorithm of SocDim is very different from both DeepWalk and node2vec since it gen-

erates the latent representation R^d from the top- d eigenvectors (Eign) of B and the modularity matrix of G . Contrary to this, both DeepWalk and node2vec first perform a random walk, obtain samples from a large graph, and then apply some form of feature learning such as SkipGram or stochastic gradient descent (SGD). In addition to this, all algorithms are relatively sensitive to the number of latent dimensions d , but the node2vec has additional function parameters p and q , which requires proper tweaking.

Examples of Application

NRL with Content Information: Since DeepWalk purely relies on structural relationships between the nodes, a logical extension to this algorithm is to incorporate content information into the feature learning process. Chang et al. (2015) propose a deep embedding algorithm called heterogeneous network embedding (HNE) that captures the complex interactions between nodes with different edge types. HNE considers both the content and the relational information of nodes to create heterogeneous embeddings.

Deep NRL Models: Some researchers use a mixture of different deep learning techniques to capture the latent representations of graphs. For instance, in Cao et al. (2016), the graph representation is learned using a three-step framework. First, it captures the structural information from weighted graphs using a random surfing model to create a probabilistic co-occurrence matrix. Second, it uses a modified pointwise mutual information (PMI) over this matrix to create a PPMI matrix, and third, a stacked denoising autoencoder is used to learn the low-dimensional vertex representations. The main advantage of this model over DeepWalk is the ability to directly obtain a probabilistic co-occurrence matrix without the need for random sampling.

NRL for Recommender Systems: Recommending suitable answers to user queries, the Q&A domain is a challenging task due to

the sparsity of links between questions and answers. However, modern Q&A websites such as Quora and Stack Exchange provide additional information in the form of user communities where a reliable user may be followed by other users. Therefore, in Fang et al. (2016), the authors use DeepWalk to extract the social information of users to solve the sparsity problem and combine it with a deep recurrent neural network which models the textual contents of questions and answers. In Chen et al. (2016), the authors propose a heterogeneous preference embedding approach (HPE) for music playlist recommendation. The input to the HPE model is a graph that is composed of users, songs, and playlists, and the output is a low-dimensional embedding of user preference vector. This preference vector is then used for recommending songs and playlists to users.

Future Directions for Research

This entry presents some recent trends in network representation learning by introducing three popular algorithms, namely, SocDim, DeepWalk, and node2vec. Future improvements and extensions to these algorithms can be categorized into the following research directions: (a) extending the model for multiple types of vertices, (b) embedding explicit domain-specific features of nodes and edges into a latent space, (c) modifying (or replacing) the feature learning process with recent techniques from deep neural networks, and (d) extending the models for signed networks.

References

- Belkin M, Niyogi P (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Advances in neural information processing systems, pp 585–591
- Brandes U, Delling D, Gaertler M, Görke R, Hoefer M, Nikoloski Z, Wagner D (2006) Maximizing modularity is hard. arXiv preprint physics/0608255
- Cao S, Lu W, Xu Q (2016) Deep neural networks for learning graph representations. In: AAAI, pp 1145–1152
- Chakrabarti D, Faloutsos C (2006) Graph mining: laws, generators, and algorithms. ACM Compu Surv (CSUR) 38(1):2
- Chang S, Han W, Tang J, Qi GJ, Aggarwal CC, Huang TS (2015) Heterogeneous network embedding via deep architectures. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 119–128
- Chen CM, Tsai MF, Lin YC, Yang YH (2016) Query-based music recommendations via preference embedding. In: Proceedings of the 10th ACM conference on recommender systems. ACM, pp 79–82
- Fang H, Wu F, Zhao Z, Duan X, Zhuang Y, Ester M (2016) Community-based question answering via heterogeneous social network learning. In: Thirtieth AAAI conference on artificial intelligence
- Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 855–864
- Jones I, Tang L, Liu H (2015) Community discovery in multi-mode networks. Springer International Publishing, pp 55–74. https://doi.org/10.1007/978-3-319-23835-7_3
- Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20(1):359–392
- Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781
- Newman ME (2006a) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74(3):036,104
- Newman ME (2006b) Modularity and community structure in networks. Proc Natl Acad Sci 103(23): 8577–8582
- Nowicki K, Snijders TAB (2001) Estimation and prediction for stochastic blockstructures. J Am Stat Assoc 96(455):1077–1087
- Perozzi B (2016) Local modeling of attributed graphs: algorithms and applications. PhD thesis, State University of New York at Stony Brook
- Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 701–710
- Rakesh V, Choo J, Reddy CK (2015) Project recommendation using heterogeneous traits in crowdfunding. In: ICWSM, pp 337–346
- Tang L, Liu H (2009) Relational learning via latent social dimensions. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 817–826
- Tang L, Liu H, Zhang J, Nazeri Z (2008) Community evolution in dynamic multi-mode networks. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 677–685

- Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: large-scale information network embedding. In: Proceedings of the 24th international conference on World Wide Web, international World Wide Web conferences steering committee, pp 1067–1077
- Tenengbaum JB, De Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500):2319–2323
- Von Luxburg U (2007) A tutorial on spectral clustering. *Stat Comput* 17(4):395–416
- Zafarani R, Abbasi MA, Liu H (2014) Social media mining: an introduction. Cambridge University Press, New York

Federated RDF Query Processing

Maribel Acosta¹, Olaf Hartig², and Juan Sequeda³

¹Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany

²Linköping University, Linköping, Sweden

³Capsenta, Austin, TX, USA

Synonyms

Federation of SPARQL endpoints; SPARQL federation

Definitions

Federated RDF query processing is concerned with querying a federation of RDF data sources where the queries are expressed using a declarative query language (typically, the RDF query language SPARQL), and the data sources are autonomous and heterogeneous. The current literature in this context assumes that the data and the data sources are semantically homogeneous, while heterogeneity occurs at the level of data formats and access protocols.

Overview

In its initial version, the SPARQL query language did not have features to explicitly express queries over a federation of RDF data sources. To support

querying such a federation without requiring the usage of specific language features, the following assumption has been made throughout the literature: The result of executing any given SPARQL query over the federation should be the same as if the query was executed over the union of all the RDF data available in all the federation members.

Formalization of SPARQL Over Federated RDF Data

To capture this assumption formally (Stolpe 2015), it is necessary to introduce a few basic concepts first: The core component of each SPARQL query is its query pattern. In the simplest case, such a query pattern is a so-called *triple pattern*, that is, a tuple $tp = (s, p, o)$ where s and o may be either a URI, an RDF literal, or a variable, respectively, and p may be either a URI or a variable. Multiple triple patterns can be combined into a set, which is called a *basic graph pattern (BGP)* and resembles the notion of a conjunctive query. In addition to such BGPs, several other features are possible in SPARQL query patterns (e.g., UNION, FILTER). For the complete syntax refer to the SPARQL specification (Harris et al. 2013).

The expected result of any given SPARQL query is defined based on an evaluation function $\llbracket \cdot \rrbracket_G$ that, given an RDF graph G , takes a SPARQL query pattern and returns a set (or multiset) of *solution mappings*, that is, partial functions that associate variables with RDF terms (i.e., URIs, literals, or blank nodes). For instance, for a triple pattern tp , the result $\llbracket tp \rrbracket_G$ contains every solution mapping whose domain is the set of all variables in tp and replacing the variables in tp according to the mapping produces an RDF triple that is in G . The evaluation function $\llbracket \cdot \rrbracket_G$ is defined recursively over the structure of all query patterns (Pérez et al. 2009).

To capture the aforementioned assumption and, thus, define SPARQL over a federation of RDF data sources (rather than over an RDF graph), it is possible to introduce another evaluation function $\llbracket \cdot \rrbracket_F$ where F is a given federation. Formally, such a federation may be captured as a tuple $F = (E, d)$ where E is the

set of all data sources in the federation and d is a function that associates every source $e \in E$ with the RDF graph that can be accessed via e . Then, the new evaluation function $\llbracket \cdot \rrbracket_F$ can be defined in terms of the standard evaluation function $\llbracket \cdot \rrbracket_G$ as follows: For every SPARQL query pattern P , $\llbracket P \rrbracket_F$ is a set of mappings such that $\llbracket P \rrbracket_F = \llbracket P \rrbracket_G$ where G is the RDF graph that is the (virtual) union of the graphs of all federation members; i.e., $G = \bigcup_{e \in E} d(e)$.

Types of RDF Data Sources

Note that the aforementioned definition is deliberately generic in what it considers to be possible federation members. Most of the current solutions for federated query processing over RDF data have focused on federations of SPARQL endpoints. Nonetheless, federations of RDF data sources may be composed of other types of sources including RDF documents or Triple Pattern Fragments.

Federation of SPARQL Endpoints. SPARQL endpoints are Web services that allow for querying RDF datasets online using the SPARQL protocol (Feigenbaum et al. 2013). Such an endpoint has a URI as its address and is, in theory, capable of evaluating any given SPARQL query over its internal RDF dataset. Therefore, SPARQL endpoints are considered RDF querying interfaces with a high expressive power. The concept of querying SPARQL endpoints has even been integrated into the latest version of the SPARQL language. That is, SPARQL 1.1 comes with an extension (Prud’hommeaux and Buil-Aranda 2013) that introduces the notion of a SERVICE clause that can be used as part of a SPARQL query pattern. This new clause has the form `SERVICE c P` where c is a URI and P is a (sub-)pattern (note that, by the SPARQL 1.1 standard, c may even be a variable, but the existing formal definition of this case (Aranda et al. 2013) is not part of the standard). Then, assuming URI c is an address of a SPARQL endpoint, the SERVICE clause means that the evaluation of P has to be delegated to this endpoint. The resulting solution mappings returned from the endpoint have to be combined

with the result of the rest of the query pattern in which the SERVICE clause is contained.

Federation of RDF Documents. This type of federation composes a Web of (Linked) Data in which the federation members are RDF documents that can be retrieved by URI dereferencing. In URI dereferencing, clients use URIs (typically found in the data) to perform HTTP requests which result in retrieving documents that contain a set of RDF triples. The process of evaluating queries over a federation of RDF documents is referred to as *Linked Data query execution* (Hartig 2013) and URI dereferencing is a core task of this process. Hartig (2012) has shown that by using this process, producing the complete query result as defined by the aforementioned evaluation function $\llbracket \cdot \rrbracket_F$ cannot be guaranteed and that there exist more restrictive approaches to define an evaluation function that can be computed completely over a federation of interlinked RDF documents.

Federation of Triple Pattern Fragments. Triple pattern fragments (TPFs) is an alternative interface for providing access to RDF data (Verborgh et al. 2016). TPF sources are able to evaluate SPARQL triple patterns over their internal RDF dataset. The result of such an evaluation is a sequence of RDF triples that match the given triple pattern; this is called a fragment. Given that some fragments may contain a large number of triples, fragments are partitioned into fragment pages that contain a fixed maximum number of triples. Then, to retrieve an entire fragment, clients must traverse the TPF pages. The expressive power of TPFs is higher than URI dereferencing, but notably lower than SPARQL endpoints. To execute an arbitrary SPARQL query using TPFs, the query engine has to decompose the query into a series of TPF requests.

Query Processing

Conceptually, the processing of queries in a federated setting can be separated into three tasks:

(i) query decomposition and source selection, (ii) query optimization, and (iii) query execution. This section provides an overview of existing approaches to tackle these tasks. To this end, the tasks are considered separately, one after another, which is also the most common approach to perform them in many RDF federation engines. However, some engines intertwine the tasks (at least partially) to account for changing data sources and runtime conditions.

Query Decomposition and Source Selection

To execute a query over a federation, the query first has to be decomposed into subqueries that are associated with data sources (federation members) selected for executing the subqueries. Essentially, the result of this step resembles a rewriting of the query into a SPARQL query with SERVICE clauses, and the objective is to achieve a complete query result with a minimum number of SERVICE clauses. An important notion in this context is the relevance of data sources for triple patterns; that is, a data source d is *relevant* for a triple pattern tp if the evaluation of tp over the data of d is nonempty; i.e., $\llbracket tp \rrbracket_{ep(d)} \neq \emptyset$.

An initial, not necessarily optimal approach to decompose a given SPARQL query is to treat each triple pattern in the query as a separate subquery and to associate each such subquery with all the data sources that are relevant for the respective triple pattern. A common improvement of such an initial decomposition is to merge subqueries that are associated with the same single data source; such merged subqueries are called *exclusive groups* (Schwarte et al. 2011).

Note that the results of the different subqueries have to be combined to produce the result of the given SPARQL query. It is possible, however, that for any of the (relevant) data sources associated with a subquery, the subquery result obtained from that source may give an empty result when combined with the results of some of the other subqueries. Hence, that subquery result does not contribute to the overall query result produced in the end. Based on this observation it is possible to prune some of the data sources associated with subqueries if there is evidence that the subquery

results from these data sources can be safely ignored without affecting the completeness of the overall query result. Source selection approaches that apply such a pruning are called *join-aware* (Saleem and Ngomo 2014).

Table 1 lists the approaches for query decomposition / source selection as proposed in the SPARQL federation literature. The main difference between these approaches (in addition to join awareness) is in the information they require about the federation members and the point at which they obtain this information. There exist approaches that rely solely on some type of *pre-populated data catalog* with (approach-specific) metadata about the federation members. Other approaches do not assume such a catalog and, instead, identify relevant federation members by querying all federation members *during the source selection process*. A third type of approaches are *hybrids* that employ a pre-populated data catalog in combination with runtime requests to the federation members. The approaches that employ a catalog (with or without

Federated RDF Query Processing, Table 1 Properties of query decomposition/source selection approaches proposed in the literature, where DC approaches rely solely on a pre-populated data catalog, RT approaches rely only on runtime requests, and HY are hybrids

Approach	Join-aware	Class
<i>Federation engine</i>		
DARQ (Quilitz and Leser 2008)	✓	DC
ADERIS (Lynden et al. 2010, 2011)	✗	DC
ANAPSID (Acosta et al. 2011)	✓	HY
SPLENDID (Görlitz and Staab 2011b)	✗	HY
FedX (Schwarte et al. 2011)	✗	RT
Prasser et al. (2012)	✓	DC
WoDQA (Akar et al. 2012)	✓	HY
LHD (Wang et al. 2013)	✗	HY
SemaGrow (Charalambidis et al. 2015)	✗	HY
Lusail (Mansour et al. 2017)	✓	RT
<i>Source selection extension</i>		
DAW (Saleem et al. 2013)	✗	HY
HiBISCuS (Saleem and Ngomo 2014)	✓	HY
Fed-DSATUR (Vidal et al. 2016)	✓	RT

additional runtime requests) can be distinguished further based on what type of metadata and statistics they capture in their catalog. Both Görlitz and Staab (2011a, Sec.5.3.2) and Oguz et al. (2015, Sec.3.1–3.2) provide a more detailed overview of existing approaches.

Query Optimization

The optimizer devises a plan that encodes how the execution of the query is carried out. The objective of the optimizer is to devise an execution plan that minimizes the cost of processing the query decompositions. In federated scenarios, both the number of intermediate results as well as the communication costs impact on the query performance.

Cost Estimation for Plans. To estimate the cost of plans in federated SPARQL query processing, Görlitz and Staab (2011b) propose a cost model that accounts for communication costs and estimates the cardinality of SPARQL expressions. Regarding the communication costs, this model assumes that all the responses from the sources require the same number of packages to be transmitted over the network and that the solution mappings have the same average size. Furthermore, this cost model relies on exact cardinalities for triple patterns with bound predicates and estimations for other variations of triple patterns, graph patterns, and joins. Nonetheless, in federated query processing where sources are autonomous, it is not always possible to collect sufficient statistics from the data sources to estimate cardinalities accurately. Therefore, some SPARQL federation engines (Acosta et al. 2011; Schwarte et al. 2011) implement heuristic-based optimization techniques that estimate the cost of subqueries based on the number of triple patterns and the number of variables in each triple pattern.

Plan Enumeration. To explore the space of query plans, optimizers of current SPARQL federation engines implement different search strategies that devise different tree plan shapes: (left-) deep plans and bushy plans. In contrast to deep plans, bushy trees are able to reduce the size of intermediate results in SPARQL queries (Vidal

et al. 2010) and enable parallel execution of independent sub-plans. One common strategy used in traditional query processing is dynamic programming, which is able to enumerate all join plans while pruning inferior plans based on the estimated cost. This strategy is implemented by SPLENDID (Görlitz and Staab 2011b) and LHD (Wang et al. 2014) to devise bushy tree plans. The main advantage of dynamic programming is that it produces the best possible plans under the assumption that cost model is accurate. However, the time and space complexity of dynamic programming is exponential, and thus this strategy does not scale for queries with a large number of subqueries. To overcome this limitation, the federated engines FedX (Schwarte et al. 2011) and ANAPSID (Acosta et al. 2011) implement heuristics to devise reasonable effective plans quickly. The heuristics implemented by FedX enumerate left-deep plans, while ANAPSID heuristics are able to devise bushy tree plans.

Further Optimization Techniques. In traditional query processing, the optimizer considers pushing down filters in the query plan which, in turn, allows for reducing the amount of intermediate results produced during query execution. Following this technique, ANAPSID (Acosta et al. 2011) implements heuristics to (i) push down filter expressions in the query plan and (ii) decide whether the filter is evaluated at the query engine or at the endpoint. In case that the filter expression contains variables that are resolved by the same source, ANAPSID ships the evaluation of the filter to the endpoint thus reducing the amount of data transmitted over the network. Other optimization techniques split composed filter expressions into sub-expressions such that each sub-expressions can be pushed down in the query plan (Görlitz and Staab 2011b).

Query Execution Techniques

During query execution, the engines evaluate the plan devised by the optimizer. In this task, the engine contacts the members of the federation identified as relevant and combines the retrieved data to produce the final query result. In federated

SPARQL query processing, physical operators and adaptive query processing techniques have been proposed to efficiently combine the intermediate results.

Physical Operators. In terms of Nested Loops Join, Görlitz and Staab (2011a) and Schwarte et al. (2011) have proposed join operators that reduce the number of requests submitted to the sources by grouping several bindings into a single request. The *distributed semi-join* (Görlitz and Staab 2011a) buffers the solution mappings in FILTER expressions using the logical connectors \wedge (when mappings contain several variables) and \vee (for grouping several mappings). In contrast, the *bound join* (Schwarte et al. 2011) groups a set of solution mappings in a single subquery using SPARQL UNION constructs. Furthermore, ANAPSID (Acosta et al. 2011) provides non-blocking join operators *agjoin* and *adjoin* that efficiently combine solution mappings able to produce results as soon as the data arrives from the endpoints.

Adaptive Query Processing. Most of the current federated SPARQL query engines follow the optimize-then-execute paradigm. In this paradigm the plan devised by the optimizer is fixed. Nonetheless, the optimize-then-execute paradigm is insufficient in federated scenarios in which the optimizer is not able to devise optimal plans due to misestimated statistics or unpredictable costs resulting from network delays or source availability. To overcome the limitations of the optimize-then-execute paradigm, some SPARQL federation engines implement adaptive query processing techniques (Deshpande et al. 2007) to adjust their execution schedulers according to the monitored runtime conditions. ANAPSID (Acosta et al. 2011) is an adaptive engine that flushes solution mappings that have matched a resource to secondary memory, in cases when the main memory becomes full. Solution mappings in secondary memory are re-considered to produce new solutions when the endpoints become blocked. The query execution techniques implemented by ANAPSID are tailored to cope

with network delays and to handle a large number of intermediate results produced by arbitrary data distributions. Another federated engine that implements adaptivity during query execution is ADERIS (Lynden et al. 2010, 2011). ADERIS is able to re-invoke the query optimizer at execution time after all the data is retrieved from the endpoints. The optimizer then changes the join ordering to devise a new efficient query plan given the current runtime conditions.

Open Research Challenges

The open research challenges can be organized in two parts. The first set of challenges are in the context of the Semantic Web. The second set of challenges extend known issues in federated query processing on relational databases.

Challenges of Using the Semantic Web as a Federation

Reasoning: RDF and SPARQL are part of a larger set of Semantic Web technologies standardized by the W3C. An important standard is the Web Ontology Language (OWL), which is a knowledge representation and reasoning language based on Description Logics. RDF systems may implement reasoners which enables inferring new data based on the input data and ontologies. A federated query processing system on RDF data coupled with OWL ontologies must be extended to support the following: (i) understanding different entailment regimes (Glimm and Ogbuji 2013) that a federation member is able to support (e.g., RDF entailment, RDFS entailment, OWL2 direct semantics entailment) and (ii) reasoning with alignments between different OWL ontologies used across different federation members. Early approaches that have taken first steps to integrate ontology alignments into federated query processing are ALOQUS (Joshi et al. 2012) and LHD-d (Wang et al. 2014).

Unboundedness: Existing approaches to federated RDF query processing assume that the complete federation is known beforehand. However,

in the context of the Semantic Web, this may not be the case. An initial set of data sources (federation members) may be known, but it is possible that all the sources are not known initially. Hence, the total number of sources that can contribute to the answer of a given query may be unbound. This unboundedness has implications in query processing, namely, in the query decomposition and source selection step.

Completeness: Recall that the result of executing any given SPARQL query over the federation should be the same as if the query was executed over the union of all the RDF data available in all the federation members. Therefore, completeness of a SPARQL query over the federation depends on knowing all the federation members. Per the unboundedness challenge (see above), if the sources are not known, then this notion of completeness should be relaxed. The implication of this challenge is to understand a tradeoff between soundness and completeness versus precision and recall of answering a query in a federation.

Dynamicity: The data of some federation members may change frequently (e.g., streams, social media). It has to be studied how such dynamicity affects the results of queries over a federation and how the dynamicity can be taken into account during federated query processing.

Challenges of Extending Federated Query Processing

Access Control: Within federated database systems, access control is a mechanism to check whether a query issued by a user is allowed or not. Given that each federation member is autonomous and has its own access control, conflicts may arise when data sources are combined in a federation. This challenge is further increased with the presence of ontologies and reasoning capabilities at the federation members. The inferred data needs to be checked with respect to the access control.

Heterogeneity: Although all the federation members are assumed to use a common data model (RDF), the members can be heterogeneous at different levels: (i) members may be using different ontologies to model data, (ii) different versions of SPARQL and different query capabilities (e.g., SPARQL endpoints vs. TPF) may have an impact on the type of queries that members are able to answer, (iii) depending on which entailment regimes are supported, a member can infer different types of answers.

Source Selection: The challenge of decomposing a query into subqueries that are associated with data sources is increased when the following additional constraints are considered. Given that data sources are autonomous, they can: (i) have different timeouts, (ii) put a limit on the number of results a query can return, (iii) support different version of SPARQL or even support subsets of SPARQL only, (iv) be of different types, such as SPARQL endpoints, RDF documents, or TPF interfaces, (v) support different entailment regimes. Therefore, a federated query processing system should take into account these constraints during the source selection.

Query Results: An important challenge in federated query processing is to associate provenance with the query results in order to understand where an answer is coming from. When sources that support reasoning such that they are able to infer new data, this adds to further explanations in the provenance of the answers.

Cross-References

- ▶ [Graph Data Models](#)
- ▶ [Graph Query Languages](#)
- ▶ [Linked Data Management](#)

References

- Acosta M, Vidal M, Lampo T, Castillo J, Ruckhaus E (2011) ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: The semantic web – ISWC

- 2011 – Proceedings of the 10th international semantic web conference, part I, Bonn, 23–27 Oct 2011, pp 18–34. https://doi.org/10.1007/978-3-642-25073-6_2
- Akar Z, Halaç TG, Ekinici EE, Dikenelli O (2012) Querying the web of interlinked datasets using VOID descriptions. In: WWW2012 workshop on linked data on the web, Lyon, 16 Apr 2012. <http://ceur-ws.org/Vol-937/lidow2012-paper-06.pdf>
- Aranda CB, Arenas M, Corcho Ó, Polleres A (2013) Federating queries in SPARQL 1.1: syntax, semantics and evaluation. *J Web Sem* 18(1):1–17. <https://doi.org/10.1016/j.websem.2012.10.001>
- Charalambidis A, Troumpoukis A, Konstantopoulos S (2015) Semagrow: optimizing federated SPARQL queries. In: Proceedings of the 11th international conference on semantic systems, SEMANTICS 2015, Vienna, 15–17 Sept 2015, pp 121–128. <https://doi.org/10.1145/2814864.2814886>
- Deshpande A, Ives ZG, Raman V (2007) Adaptive query processing. *Found Trends Databases* 1(1):1–140. <https://doi.org/10.1561/1900000001>
- Feigenbaum L, Williams GT, Clark KG, Torres E (2013) SPARQL 1.1 protocol. W3C recommendation. Online at <https://www.w3.org/TR/sparql11-protocol/>
- Glimm B, Ogbuji C (2013) SPARQL 1.1 entailment regimes. W3C recommendation. Online at <https://www.w3.org/TR/sparql11-entailment/>
- Görlitz O, Staab S (2011a) Federated data management and query optimization for linked open data. In: New directions in web data management 1. Springer, pp 109–137. https://doi.org/10.1007/978-3-642-17551-0_5
- Görlitz O, Staab S (2011b) SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Proceedings of the second international workshop on consuming linked data (COLD2011), Bonn, 23 Oct 2011. http://ceur-ws.org/Vol-782/GoerlitzAndStaab_COLD2011.pdf
- Harris S, Seaborne A, Prud'hommeaux E (2013) SPARQL 1.1 query language. W3C recommendation. Online at <http://www.w3.org/TR/sparql11-query/>
- Hartig O (2012) SPARQL for a web of linked data: semantics and computability. In: The semantic web: research and applications – Proceedings of the 9th extended semantic web conference, ESWC 2012, Heraklion, Crete, 27–31 May 2012, pp 8–23. https://doi.org/10.1007/978-3-642-30284-8_8
- Hartig O (2013) An overview on execution strategies for linked data queries. *Datenbank-Spektrum* 13(2):89–99. <https://doi.org/10.1007/s13222-013-0122-1>
- Joshi AK, Jain P, Hitzler P, Yeh PZ, Verma K, Sheth AP, Damova M (2012) Alignment-based querying of linked open data. In: On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10–14, 2012. Proceedings, Part II, pp 807–824. https://doi.org/10.1007/978-3-642-33615-7_25
- Lynden SJ, Kojima I, Matono A, Tanimura Y (2010) Adaptive integration of distributed semantic web data. In: Databases in networked information systems, proceedings 6th international workshop, DNIS 2010, Aizu-Wakamatsu, 29–31 Mar 2010. pp 174–193. https://doi.org/10.1007/978-3-642-12038-1_12
- Lynden SJ, Kojima I, Matono A, Tanimura Y (2011) ADERIS: an adaptive query processor for joining federated SPARQL endpoints. In: On the move to meaningful internet systems: OTM 2011 – Proceedings of the confederated international conferences: CoopIS, DOA-SVI, and ODBASE 2011, part II, Hersonissos, Crete, 17–21 Oct 2011, pp 808–817. https://doi.org/10.1007/978-3-642-25106-1_28
- Mansour E, Abdelaziz I, Ouzzani M, Aboulnaga A, Kalnis P (2017) A demonstration of lusail: querying linked data at scale. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD conference 2017, Chicago, 14–19 May 2017, pp 1603–1606. <https://doi.org/10.1145/3035918.3058731>
- Oguz D, Ergenc B, Yin S, Dikenelli O, Hameurlain A (2015) Federated query processing on linked data: a qualitative survey and open challenges. *Knowl Eng Rev* 30(5):545–563. <https://doi.org/10.1017/S026988915000107>
- Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of SPARQL. *ACM Trans Database Syst* 34(3):16:1–16:45. <https://doi.org/10.1145/1567274.1567278>
- Prasser F, Kemper A, Kuhn KA (2012) Efficient distributed query processing for autonomous RDF databases. In: Proceedings of the 15th international conference on extending database technology, EDBT'12, Berlin, 27–30 Mar 2012, pp 372–383. <https://doi.org/10.1145/2247596.2247640>
- Prud'hommeaux E, Buil-Aranda C (2013) SPARQL 1.1 federated query. W3C recommendation. Online at <https://www.w3.org/TR/sparql11-federated-query/>
- Quilitz B, Leser U (2008) Querying distributed RDF data sources with SPARQL. In: The semantic web: research and applications, proceedings of the 5th European semantic web conference, ESWC 2008, Tenerife, Canary Islands, 1–5 June 2008, pp 524–538. https://doi.org/10.1007/978-3-540-68234-9_39
- Saleem M, Ngomo AN (2014) Hibiscus: hypergraph-based source selection for SPARQL endpoint federation. In: The semantic web: trends and challenges – proceedings of the 11th international conference, ESWC 2014, Anissaras, Crete, 25–29 May 2014, pp 176–191. https://doi.org/10.1007/978-3-319-07443-6_13
- Saleem M, Ngomo AN, Parreira JX, Deus HF, Hauswirth M (2013) DAW: duplicate-aware federated query processing over the web of data. In: The semantic web – ISWC 2013 – proceedings of the 12th international semantic web conference, part I, Sydney, 21–25 Oct 2013, pp 574–590. https://doi.org/10.1007/978-3-642-41335-3_36
- Schwarze A, Haase P, Hose K, Schenkel R, Schmidt M (2011) Fedx: optimization techniques for federated query processing on linked data. In: The semantic web – ISWC 2011 – proceedings of the 10th international

- semantic web conference, part I, Bonn, 23–27 Oct 2011, pp 601–616. https://doi.org/10.1007/978-3-642-25073-6_38
- Stolpe A (2015) A logical characterisation of SPARQL federation. *Semantic Web* 6(6):565–584. <https://doi.org/10.3233/SW-140160>
- Verborgh R, Sande MV, Hartig O, Herwegen JV, Vocht LD, Meester BD, Haesendonck G, Colpaert P (2016) Triple pattern fragments: a low-cost knowledge graph interface for the web. *J Web Sem* 37–38:184–206. <https://doi.org/10.1016/j.websem.2016.03.003>
- Vidal M, Ruckhaus E, Lampo T, Martínez A, Sierra J, Polleres A (2010) Efficiently joining group patterns in SPARQL queries. In: The semantic web: research and applications, proceedings of the 7th extended semantic web conference, part I, ESWC 2010, Heraklion, Crete, 30 May–3 June, 2010, pp 228–242. https://doi.org/10.1007/978-3-642-13486-9_16
- Vidal M, Castillo S, Acosta M, Montoya G, Palma G (2016) On the selection of SPARQL endpoints to efficiently execute federated SPARQL queries. *Trans Large-Scale Data- Knowl Cent Syst* 25:109–149. https://doi.org/10.1007/978-3-662-49534-6_4
- Wang X, Tiropinis T, Davis HC (2013) LHD: optimising linked data query processing using parallelisation. In: Proceedings of the WWW2013 workshop on linked data on the web, Rio de Janeiro, 14 May 2013. <http://ceur-ws.org/Vol-996/papers/lidow2013-paper-06.pdf>
- Wang X, Tiropinis T, Davis HC (2014) Optimising linked data queries in the presence of co-reference. In: The semantic web: trends and challenges – proceedings of the 11th international conference, ESWC 2014, Anissaras, Crete, 25–29 May 2014, pp 442–456. https://doi.org/10.1007/978-3-319-07443-6_30

Federation of SPARQL Endpoints

- ▶ Federated RDF Query Processing

Figure of Merit

- ▶ Metrics for Big Data Benchmarks

Flood Detection

- ▶ Flood Detection Using Social Media Big Data Streams

Flood Detection Using Social Media Big Data Streams

Muhammad Hanif, Muhammad Atif Tahir,
Muhammad Rafi, and Furqan Shaikh

School of Computer Science, National
University of Computer and Emerging Sciences,
Karachi, Pakistan

F

Synonyms

Deep learning; Flood detection; Machine learning; Metadata

Definitions

Water is one of the vital substances for life, which occupies two-thirds of the earth's surface. For the survival of living beings, water management is critically important. One of the most common sources of its heavy wastage is flood. Conventional methods of disaster management have improved at large extent. Similarly, flood prediction, localization, and recovery management mechanism have been upgraded from manual reporting to advanced sensor-based intelligent systems. Additionally, data from social media big data streams appeared as a main source of information flow for ordinary events as well as for emergency situations. Millions of images along with text streams, audio clips, and videos are uploaded from different corners of the world. Data from social media is then processed to gain various socioeconomic benefits including weather prediction, disaster awareness, and so on. The aim of this chapter is to discuss and evaluate state-of-the-art computer vision and natural language techniques for flood detection using social media big data streams.

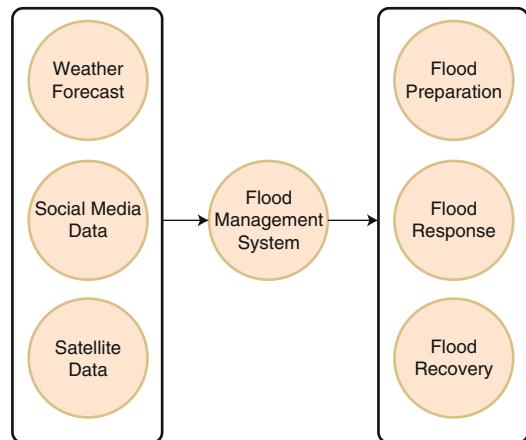
Overview

This chapter provides comprehensive review of the state-of-the-art natural language processing

(NLP), machine learning (ML), and deep learning (DL) technologies for flood detection. This chapter also summarizes the performances of several ensemble techniques where floods are predicted using the combination of meta and visual data.

Introduction

Hydroinformatics helps to effectively manage water resources of the world, by utilizing information and communication technology (Abbott et al. 1991). Conventional methods of hydroinformatics have entirely changed with the advent of technological advancement. Traditionally, gauging machines were used to manage flood detection by examining the level of water. Floods are usually caused by overflowing of river, more rainfall than available drainage system, sudden breach of protection made against it, or unexpected opening of hydropower (Jongman et al. 2015). In case of weak protective measures, floods may create dangerous situation for living beings. Mostly, valuable economical resources are also highly affected by floods. The strong defensive measures are directly linked with the information regarding the flood. The strong flood response system performs its function on the basis of the data collected from different sources including weather forecast, social media contents, and data collected from satellite system, as shown in Fig. 1. The information helps to indicate time, location, causes, and impact of the flood. Flood management system makes preparation as a response to any possibility of the flood. The response includes relief during the event. While recovery starts after the end of the emergency, to overcome the situation (Jiang et al. 2013; Chen and Han 2016; Jongman et al. 2015), there are various resources, from which useful information is generated to overcome the situation, which includes social media contents, satellite data, and weather management data. On the basis of these information resources, flood management system performs the responsibilities of preparation, response, and recovery, as shown in Fig. 1.



Flood Detection Using Social Media Big Data Streams, Fig. 1 Flood management system

The omnipresent nature of Internet has provided social interactivity pathway to the users. Moreover, social media has provided the opportunity of rapid interaction to its users and effortless sharing of text, images, audio, and video as well as any of their combination. Various social media giants are competing to attract maximum users, and Facebook tops the list. As per an official announcement, Facebook has achieved the target of 2 billion users by June, 2017. Big data collected from social media can be used for the well-being of the society and different situational awareness analysis. Specially, people use social media during emergency situations like storms and floods, to spread awareness among people of the surrounding vicinity, authorities, and humanitarian organization so that the situation may be tackled appropriately. Along with the data from social media, satellites are also used for the prediction and detection of flooding events (Pohl et al. 2012, 2016; Fohringer et al. 2015).

The massive quantity of disaster-related information is regularly collected from social media and satellite. These huge datasets require advanced storage and processing technologies. The term “big data” is used to denote complex and huge datasets, which cannot be captured, stored, and processed by normal computing technologies including big data analytics and cloud computing (Shamsi et al. 2013). The big data techniques

create patterns by exploring huge amount of data. This motivates usage of big data techniques in disaster management systems, especially in flood management (Eilander et al. 2016; Chen and Han 2016). The main aim of this chapter is to discuss the existing challenges for flood detection using social media big data streams. This chapter will also provide comprehensive review of the state-of-the-art natural language processing (NLP), machine learning (ML), and deep learning (DL) technologies for flood detection.

Literature Review

Most awful situations are faced by the countries with scarcity of resources. They are unable to handle floods properly. Consequently, humanitarian organizations play their role to handle the situation. Moreover, countries having a larger area are also facing troubles in protecting larger region from flooding events. Also, there is a lack of proper intercommunication mechanism between countries to share the experiences and knowledge regarding flood management. Flooding events in Pakistan and Philippines are analyzed to explore the timing of occurrence of flood, mapping of its location, and understanding of its impact (Jongman et al. 2015). Data has been gathered by using organizations of disaster response, Global Flood Detection System (GFDS) satellite signals, and information from Twitter accounts. It was observed that GFDS satellite information is appropriate to produce better results in case of larger floods while less appropriate for flood with smaller duration. It also has a drawback, as it produces errors while calculating data in coastal areas, so it is more suitable to be used in regions of delta or islands. Moreover, Twitter information analysis for flood exploration can produce fruitful results in handling flood of any size. However, it has challenges of finding the appropriate location of event. Additionally, Twitter population is limited in rural areas, which hinders the accurate and timely information (Jongman et al. 2015).

In an experimental research study (Imran et al. 2016), a huge corpus of 52 million Twitter messages has been gathered during the period

of 2013–2015. Dataset was collected from different parts of the world, during 19 different incidents of crises including earthquakes, floods, landslides, typhoons, volcanos, and airline accidents. Artificial Intelligence for Disaster Response (AIDR), an open source platform, is used for the collection and classification of the messages. Additionally, the platform provides easy method to collect tweets from specific regions with particular keywords. Messages are categorized according to their contents as injured and dead people, missing people, damaged infrastructure, volunteer services, donations, sympathy, and emotional support. Before classification, stop-words and uniform resource locators (URLs) are removed from the data; both of them have no or less role in classification. Moreover, Lovins stemmer is used, which removes the derivational part of the word to make it more clear like “flooding” will be replaced by “flood.” Unigram and bigram words are defined as features. However, the information gained is used to select the top 1000 effective features. After the selection of features, multiclass categorization is performed by using support vector machine (SVM), random forest (RF), and Naïve Bayes (NB) algorithms. Evaluation of trained models is performed by tenfold cross validation, and largest word embedding on the topic of crisis management has been developed (Imran et al. 2016).

Another research (Nguyen et al. 2016) has explored stochastic gradient descent and proposed online algorithm using deep neural network (DDN). This algorithm is aimed to perform identification and classification of significant tweets. For identification, informative effectiveness of each tweet is investigated, and as a result, two categories were formed which were tagged as “informative” and “non-informative.” Informative tweets contains indication regarding the category of destruction due to disaster. Furthermore, the tweets are categorized according to their contents, as tweets containing an indication of loss of infrastructure are placed in a different category, while tweets with information regarding deceased and injured persons are placed in a separate category. Convolutional neural network

Flood Detection Using Social Media Big Data Streams, Table 1 Some public datasets for disaster event

Media	Samples	Algorithm	Duration	Disaster event
Twitter, satellite (Jongman et al. 2015)	7.6 million	–	2014–2015	Flood
Twitter (Imran et al. 2016)	52 million	NB, RF, SVM	2013–2015	Flood, earthquake
Twitter (Nguyen et al. 2016)	0.11 million	DNN	2015	Earthquake

(CNN)-based model is trained and tested for multiple six classes, including “donation and volunteering,” “sympathy and support,” and so on (Nguyen et al. 2016). Some public datasets for disaster events are shown in Table 1.

The severity of flood is diverse for different parts of the world. Few countries are significantly affected, as in Malaysia 40% of the total damage is caused by floods. The flood prone region of Terengganu, Malaysia, has been mapped by implementing support vector machine (SVM). The flood event took place in this area in November, 2009. Flood locations were detected by the method of texture analysis, and 181 flood-affected locations were recorded. The quantity of 70% locations was selected as training set and remaining 30% as testing set. Availability of flood or its absence is verified by binary classification with the help of various mathematical functions called kernels. This approach has used sigmoid (SIG), radial basis function (RBF), linear (LN), and polynomial (PL) kernel functions. Results were compared with the popular method of frequency ratio; it is found that SVM linear and SVM polynomial achieved success rate of 84.31% and 81.89%, respectively, while 72.48% by frequency with ratio method (Tehrany et al. 2015).

Efficient Natural Language Processing (NLP) Techniques for Flood Detection Using Social Media Big Data Text Streams

Natural language processing (NLP) plays a vital role in progressing artificial intelligence (AI) research in many practical areas of life. There are many several challenges in the NLP realm to be resolved such as natural language generation, natural language understanding, dialog

systems, speech generation, and recognition systems to name a few. Major fields that support NLP research are computer science, artificial intelligence, and computational linguistics. On the Internet 80% of user-generated contents are in textual form, and the potential of this huge knowledge resource is still to be explored. Internet with its recent advancement in real-time interactive user-/social group-generated textual content and readership offers new potential applications in the area of real-time event-based information processing and management. The NLP-based applications generally have a preprocessing pipeline for processing of textual features into more meaningful units. There can be parsing of lexeme from text, stemming/lemmatization, statistical, or counting of features. During the US presidential election of 2016, Twitter proved to be the largest and powerful source of breaking news, with 40 million election-related tweets posted by 10 PM on election day. Social media platforms, including Twitter, have been used and analyzed for use in disasters and emergency situations. Kaminska and Rutten (2014) identified the main uses of social media across the four pillars of the disaster management cycle being prevention and mitigation, preparedness, response, and recovery. Following are the three main areas where these platforms can be very helpful (Dufty et al. 2016).

- public information
- situational awareness
- community empowerment and engagement

Natural language processing (NLP) is a significant research area, which aims to achieve computational language processing mechanism, similar to human beings. Various desired tasks may be accomplished through NLP, including translation of text from one language to another,

paraphrasing, summarization, informational retrieval, and so on. Nowadays, social media contents are a huge source of information; hence appropriate processing could reveal valuable outcomes. The text available in social media is in raw format and needs preprocessing. Various techniques are used for preprocessing including stemming and lemmatization. Stemming is a process of finding the root or the main word from its derived word. For example, if stemming is applied on word flooding and storming, they will be reverted to their root words as “flood” and “storm,” respectively. Lemmatization removes unnecessary part of the word and provides root word, which must be a dictionary word with proper meaning (Bramer 2007; Eilander et al. 2016).

Challenges in NLP

Social Media Challenges

The information available in the form of social media contents is unreliable in various aspects. The creators of the information are not experts of the field; this inexperience may result in wrong judgment. Moreover, there are no restrictions on the usage of slang language. As the word flood may be used to indicate a huge quantity of people or things arriving at some place. Additionally, information mentioned on social media could indicate past experiences, rather than current situations (Eilander et al. 2016).

Challenges of Using Tweets for Disaster Response System

Twitter is an important part of social media, where the user may upload text up to 140 characters. Usage of data collected from tweets for crisis management faces various challenges due to the informal and restrictive nature of Twitter. The major drawback is unintentional spelling mistakes due to negligence of user. Additionally, due to the restriction of fewer words, the user may intentionally use formal or informal abbreviations of the words as; “Govt.” or “gov” instead of “government” and “2morrow” instead of “tomorrow.” Sometimes, the user faces inputting the whole idea in a restrictive space and removes spaces between words and write them

together, as “floodingEvent” instead of “flooding event,” which creates ambiguity (Imran et al. 2016; Nguyen et al. 2016).

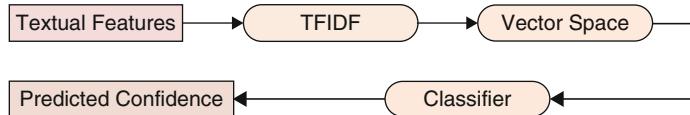
Machine Learning and NLP

Naïve Bayes (NB) and Multinomial Naïve Bayes

Naïve Bayes algorithm is highly simple and efficient. It uses probability theory for predicting the class for the given instance. The algorithm assumes that attributes of the instances are independent from each other (Bramer 2007). The process of text classification considers each word appearing in the document as a separate attribute; hence Naïve Bayes is an appropriate option to process them. Initially, multivariate Bernoulli model of Naïve Bayes classifier is proposed. It creates a binary vector form of document; each instance of the vector indicates availability or unavailability of particular word in that document. This model is more applicable to text classification with constant number of attributes. It lacks in counting the frequency of words appearing in the document. To address this issue, multinomial Naïve Bayes (MNB) is introduced. It counts the frequency of each word appearing in the document (Jiang et al. 2013).

Deep Neural Network (DNN)

In contrast to manual feature extraction, deep neural network (DNN) automatically extracts the abstract features from the data, which helps in the construction of effective classes. The network is mostly trained by using online algorithms, which is a favorable feature to be used for disastrous situations. The adaptive nature of deep neural network (DNN) attracts its usage in crisis management. The model can be trained earlier, while in case of a newly available dataset, it can work fine in the classification of newly arrived datasets, which is ideal for a crisis situation where real-time data needs rapid processing (Nguyen et al. 2016). Figure 2 shows the flow diagram for metadata feature processing using TF-IDF. Naive Bayes and support vector machines are the most popular classifiers in NLP Systems.



Flood Detection Using Social Media Big Data Streams, Fig. 2 Data flow diagram for metadata feature processing using TF-IDF

Flood Detection Using Social Media Big Data Streams

Table 2 Some state-of-the-art NLP techniques for flood detection using social media text data stream

Technique	Dataset	Evaluation measure	Performance
Relation networks (Nogueira 2017)	MediaEval	Average precision (Top 480)	77.0
Logistic regression, multinomial Naïve Bayes, random forest (Tkachenko et al. 2017)	MediaEval	Average precision (Top 480)	66.0
Multinomial Naïve Bayes, SR-KDA (Hanif et al. 2017)	MediaEval	Average precision (Top 480)	65.0
Random forest, decision tree, kernel SVM, k-NN, logistic regression (Ramaswamy et al. 2017)	Bengaluru environmental dataset	Accuracy	99.0%
Linear SVM, linear regression, XGBoost regression (Ramaswamy et al. 2017)	SemEval	R ² Score, cosine Similarity	0.63
SVM, nearest neighbor (Lamovc et al. 2013)	Centre of excellence space-SI	Accuracy	83.0%
Latent semantic analysis (LSA) (Basnyat et al. 2017)	Twitter	Accuracy	94.0%

State-of-the-Art NLP Systems for Flood Detection Using Big Data Streams

Recently, various novel systems have been proposed to detect flood through text analysis of social media big data streams (Bischke et al. 2017; Hanif et al. 2017; Ahmad et al. 2017b,a; Avgerinakis et al. 2017; Nogueira 2017). Most of these approaches are evaluated on standard public dataset from Multimedia Satellite Task at MediaEval, 2017. This dataset consists of metadata of images uploaded by the social media users, which may or may not contain evidence of flood. This metadata contains various fields including relevant description of individual image, user tags, title, date on which the image was uploaded, and the device through which it was

captured. The main objective is to extract and fuse the content of events which are present in satellite imagery and social media (Bischke et al. 2017). A lot of different and diverse techniques are reported including metadata features such as term frequency-inverse document frequency (TF-IDF) (Bischke et al. 2017; Hanif et al. 2017) and word/text embeddings from a large collection of titles, descriptions, and user tags (Tkachenko et al. 2017). Among them, traditional features like TF-IDF are quite useful since it integrates the concept of term frequency that counts the number of occurrences of each semantically similar concepts (“flood,” “river,” “damage”) in the given user tags. Table 2 summarizes the performances of several NLP-based techniques.

Flood Detection from Social Media Visual Big Data Stream: Machine Learning and Deep Learning Techniques

Visual images available on social media can be effectively used for flood management system. Figure 3 shows some sample images from MediaEval 2017 satellite task. Over the years, various computer vision techniques have been investigated to extract local/global visual features from these images (Bischke et al. 2017). These features include AutoColorCorrelogram, EdgeHistogram, Color and Edge Directivity Descriptor (CEDD), Color Layout, Fuzzy Color and Texture Histogram (FCTH), Joint Composite Descriptor (JCD), Gabor, ScalableColor, Tamura, SIFT, Colour SIFT, and local binary patterns (LBP). Machine learning techniques such as support vector machine (SVM), decision tree (DT), and kernel discriminant analysis using spectral

regression (SR-KDA) are then used to distinguish between flood and non-flood images.

Recently, deep learning is quite successful in image classification tasks. Deep learning is an advanced field of machine learning which portrays human brain and produces similar neural network for execution of tasks. It executes input through hierarchy of multiple layers to produce proper information. In contrast to machine learning, features are automatically extracted from the provided input which saves a lot of human effort and computing resources (LeCun et al. 2015). Due to real-time feature extraction and processing according to the required task, more computation is required; that is why graphical processing units (GPUs) are used for efficient results. For improved accuracy, neural networks require large amount of data, so that exact features may be predicted clearly. Deep learning has various applications in different fields including natural language processing (NLP), com-



Flood Detection Using Social Media Big Data Streams, Fig. 3 Some images from MediaEval 2017 satellite task (Bischke et al. 2017)

puter vision, audio processing, and so on. Especially in computer vision, deep learning has produced exceptional improvements to improve various dimensions including image segmentation, compression, localization, transformation, sharpening, completion, colorization, and prediction. Table 3 summarizes the performances of several ML- and DL-based techniques.

Transfer Learning

The effectiveness of deep network is directly related with the availability of training data; greater amount of training data reveals much accurate output. But, the strong training requires a huge quantity of resources. Transfer learning is proposed to resolve the issue of effective training, which does not require training of neural network from scratch. Instead, the mechanism performs little modification in extensively trained network, which is trained on large datasets and strong machines. These pre-trained neural networks can be used in a similar or different domain.

Importance of Deep Learning (DL)

Algorithms in Event Prediction

During disaster and crises, accurate situational awareness and rapid communication are

important to properly manage the situation. Conventional algorithms perform their functionality according to the defined steps, and they are inappropriate to manage crises. On contrary, deep learning algorithms perform their task according to input provided to them, and they are more appropriate to respond contextual awareness, so that accelerated action may be performed. ImageNet dataset: ImageNet is a dataset, which contains 15 million labeled images, categorized into about 22,000 categories (Jiang et al. 2013). Yearly, a contest is organized to produce better results by using ImageNet. The winner of that competition reveals new model, including AlexNet, QuocNet, Inception or GoogLeNet and BN-Inception-v2, and Inception version 3. These pre-trained models are released to be executed on machines having low computational and storage resources. One or more layers of these networks are altered according to domain, and then the network is retrained accordingly. This retrained network produce better efficiency due to the strong training it acquires from ImageNet dataset. The Inception version 3 was introduced in 2016 (Chen and Han 2016), with boosted performance on ILSVRC 2012 classification benchmark. In comparison with the state-of-

Flood Detection Using Social Media Big Data Streams, Table 3 Some state-of-the-art machine learning and deep learning techniques for flood detection using social media visual data stream

Technique	Dataset	Evaluation measure	Performance
Convolutional neural network, AlexNet DL model (Ahmad et al. 2017b)	MediaEval	Average precision (Top 480)	86.0
GoogleNet DL model (Nogueira 2017)	MediaEval	Average precision (Top 480)	74.0
Global/local features, SR-KDA (Hanif et al. 2017)	MediaEval	Average precision (Top 480)	64.0
Ensemble CNNL,STM, and relation networks (Santoro et al. 2017)	Clever	Accuracy	96.4%
ANN, SOM maps (Kussul et al. 2008)	ERS-2/SAR Ukraine	Accuracy	85.40%
Adaboost (CART, SVM, RF, DNN) (Coltin et al. 2016)	MODIS satellites (Sava River)	Precision/recall	95%, 98%
Discrete cosine transfer (DCT) (Basnyat et al. 2017)	Twitter	Accuracy	67.0%

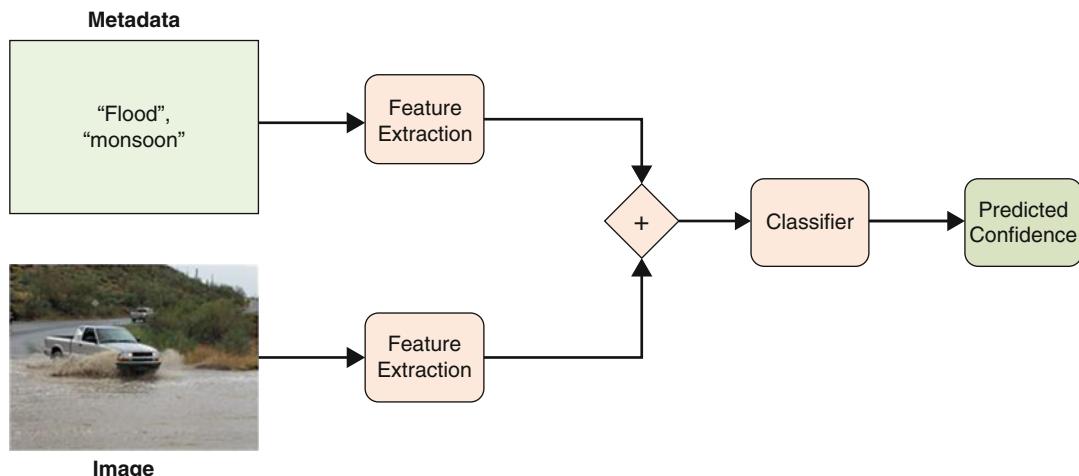
the-art results (Nguyen et al. 2016), it has cut error by 25%, with six times cheaper computing resource with five times less parameters.

Combining Knowledge Using Fusion of Meta- and Visual Data

The performance of flood detection can be improved by using both image uploaded on social media and its metadata. The combination utilizes features of text as well as images; this enhances the effectiveness of the results. Figure 4 shows an example showing fusion of metadata and visual data. Table 4 summarizes the performances of several ensemble techniques. In these techniques, floods are predicted using the combination of meta- and visual data.

Conclusion

The idea of harnessing the social media for emergency management is quite effective. The fundamental rules of social media like instantaneous, cost-effective, transparent, and intentional are very much related to the objectives of emergency management which are collaborative, coordinated, integrated, progressive, and professional. Most emergency agencies around the world now use social media alongside the conventional platforms like newspaper, television, and community channels to establish the process of warning, safety, response, and recovery from the emergency situation. In this paper, we briefly explore the flood detection systems using social media big data streams. Re-



Flood Detection Using Social Media Big Data Streams, Fig. 4 Data flow diagram showing fusion of meta- and visual data

Flood Detection Using Social Media Big Data Streams, Table 4 Some state-of-the-art ensemble techniques for flood detection using fusion of meta- and visual data

Technique	Dataset	Evaluation measure	Performance
Relation networks and CNN (Nogueira 2017)	MediaEval	Average precision (Top 480)	95.0
Logistic regression classifier, multinomial Naïve Bayes, random forest (Tkachenko et al. 2017)	MediaEval	Average precision (Top 480)	66.0
TFIDF, global/ local visual features, SR-KDA (Hanif et al. 2017)	MediaEval	Average precision (Top 480)	64.0

cently, there are several datasets shared across the world from different countries/agencies for flood-related emergency situations. Moreover, there are integrated systems/frameworks for emergency reporting. These systems use several state-of-the-art machine learning approaches to determine the response in the critical situations. The integration of image and social media textual data deals the right mixture, for effective predictions and monitoring. The size of social media data, its integrations, filtration, and categorization to the relevant situation offer many challenges. On the other hand, its effective use in crisis detection, resilience, learning, and capacity building for real and online analysis cannot be denied.

Cross-References

- [Big Data Analysis for Social Good](#)
- [Big Data in Social Networks](#)

References

- Abbott MB et al (1991) Hydroinformatics: information technology and the aquatic environment. Avebury Technical
- Ahmad K, Konstantin P, Riegler M, Conci N, Holversen P (2017a) CNN and GAN based satellite and social media data fusion for disaster detection. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Ahmad S, Ahmad K, Ahmad N, Conci N (2017b) Convolutional neural networks for disaster images retrieval. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Avgerinakis K, Mountzidou A, Andreadis S, Michail E, Gialampoukidis I, Vrochidis S, Kompatsiaris I (2017) Visual and textual analysis of social media and satellite images for flood detection@ multimedia satellite task mediaeval 2017. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Basnyat B, Anam A, Singh N, Gangopadhyay A, Roy N (2017) Analyzing social media texts and images to assess the impact of flash floods in cities. In: 2017 IEEE international conference on smart computing (SMARTCOMP). IEEE, pp 1–6
- Bischke B, Bhardwaj P, Gautam A, Helber P, Borth D, Dengel A (2017) Detection of flooding events in social multimedia and satellite imagery using deep neural networks. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Bramer M (2007) Principles of data mining, vol 180. Springer, London
- Chen Y, Han D (2016) Big data and hydroinformatics. *J Hydroinf* 18(4):599–614
- Coltin B, McMichael S, Smith T, Fong T (2016) Automatic boosted flood mapping from satellite data. *Int J Remote Sens* 37(5):993–1015
- Dufty N et al (2016) Twitter turns ten: its use to date in disaster management. *Aust J Emerg Manag* 31(2):50
- Eilander D, Trambauer P, Wagemaker J, van Loenen A (2016) Harvesting social media for generation of near real-time flood maps. *Procedia Eng* 154:176–183
- Fohringer J, Dransch D, Kreibich H, Schröter K (2015) Social media as an information source for rapid flood inundation mapping. *Nat Hazards Earth Syst Sci* 15(12):2725–2738
- Hanif M, Tahir MA, Khan M, Rafi M (2017) Flood detection using social media data and spectral regression based kernel discriminant analysis. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Imran M, Mitra P, Castillo C (2016) Twitter as a lifeline: human-annotated twitter corpora for NLP of crisis-related messages. arXiv preprint arXiv:160505894
- Jiang L, Cai Z, Zhang H, Wang D (2013) Naive bayes text classifiers: a locally weighted learning approach. *J Exp Theor Artif Intell* 25(2):273–286
- Jongman B, Wagemaker J, Romero BR, de Perez EC (2015) Early flood detection for rapid humanitarian response: harnessing near real-time satellite and twitter signals. *ISPRS Int J Geo-Inf* 4(4):2246–2266
- Kussul N, Shelestov A, Skakun S (2008) Grid system for flood extent extraction from satellite images. *Earth Sci Inf* 1(3):105
- Lamovec P, Matjaz M, Ostir K (2013) Detection of flooded areas using machine learning techniques: case study of the Ljubljana moor floods in 2010. *Dis Adv* 6:4–11
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444
- Nguyen DT, Joty S, Imran M, Sajjad H, Mitra P (2016) Applications of online deep learning for crisis response using social media information. arXiv preprint arXiv:161001030
- Nogueira K (2017) Data-driven flood detection using neural networks. In: Proceedings of the MediaEval 2017 workshop, Dublin
- Pohl D, Bouchachia A, Hellwagner H (2012) Automatic sub-event detection in emergency management using social media. In: Proceedings of the 21st international conference on World Wide Web. ACM, pp 683–686
- Pohl D, Bouchachia A, Hellwagner H (2016) Online indexing and clustering of social media data for emergency management. *Neurocomputing* 172: 168–179
- Ramaswamy B, Likith Ponnanna PB, Vishruth K (2017) Urban flood forecast using machine learning on real time sensor data. *Trans Mach Learn Artif Intell* 5(5):69
- Santoro A, Raposo D, Barrett DG, Malinowski M, Pascanu R, Battaglia P, Lillicrap T (2017) A simple neural

- network module for relational reasoning. arXiv preprint arXiv:170601427
- Shamsi J, Khojaye MA, Qasmi MA (2013) Data-intensive cloud computing: requirements, expectations, challenges, and solutions. *J Grid Comput* 11(2): 281–310
- Tehrany MS, Pradhan B, Mansor S, Ahmad N (2015) Flood susceptibility assessment using GIS-based support vector machine model with different kernel types. *Catena* 125:91–101
- Tkachenko N, Zubiaga A, Procter RN (2017) Wisc at mediaeval 2017: multimedia satellite task. In: Proceedings of the MediaEval 2017 workshop

Forecasting Business Process Future

► Predictive Business Process Monitoring

Framework-Based Scale-Out RDF Systems

Marcin Wylot¹ and Sherif Sakr²

¹ODS, TU Berlin, Berlin, Germany

²Institute of Computer Science, University of Tartu, Tartu, Estonia

Synonyms

Hadoop-based RDF query processors;
Spark-based RDF query processors

Definitions

RDF, the Resource Description Framework, has been recognized as a de facto standard to describe resources in a semi-structured manner. In particular, RDF is a graph-based format which allows to define named links between resources in the form of triples *subject*, *predicate*, *object*, also called statements. A statement expresses a relationship (defined by a predicate) between resources (*subject* and *object*). The relationship is always from

subject to object (it is directional). The same resource can be used in multiple triples playing the same or different roles, e.g., it can be used as a subject in one triple, as well as a predicate or an object in another one. This ability enables definition of multiple connections between the triples, hence creation of a connected graph of data. Such graph can be represented as nodes that stands for the resources and edges capturing the relationships between the nodes. RDF has been widely adopted in several application domains. For example, a growing number of ontologies and knowledge bases storing millions to billions of facts (e.g., *DBpedia*, *Probbase*, and *Wikidata*) are now publicly available (Poggi et al. 2008). In addition, key search engines like Google and Bing are providing increasing support for RDF. This entry gives an overview of various systems that have been designed to exploit the Hadoop and Spark frameworks for building scale-out RDF processing engines.

Overview

In 2004, Google made a seminal contribution to the big data community by introducing the **MapReduce** model (Dean and Ghemawat 2004). It is a simple and powerful programming model for developing scalable parallel applications that can process large datasets across multiple machines. In particular, MapReduce makes programmers think in a *data-centric* style allowing them to concentrate on dataset transformation, while MapReduce takes care of the distributed execution and fault tolerance details, transparently (Sakr et al. 2013). The Apache Hadoop project has been introduced by Yahoo! as an open-source framework that implemented the concepts of MapReduce. However, one of the main limitations of the Hadoop framework is that it requires that the entire output of each map and reduce task to be materialized into a local file on the Hadoop Distributed File System (HDFS) before it can be consumed by the next stage. This materialization step allows for the implementation of a simple and elegant checkpoint/restart fault-tolerant mechanism;

however, it dramatically harms the system performance. The **Spark** project was developed as a big data processing framework that tackles this challenge (Zaharia et al. 2010). In particular, the fundamental programming abstraction of Spark is called *Resilient Distributed Dataset* (RDD) (Zaharia et al. 2010) which represents a logical collection of data partitioned across machines. In principle, having RDD as an in-memory data structure enables Spark to exploit its functional programming paradigm by allowing user programs to load data into a cluster's memory and query it repeatedly. In addition, users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. Therefore, Spark takes the concepts of Hadoop to the next level by loading the data in distributed memory and relying on cheaper shuffle during the data processing (Sakr 2016).

Examples of Systems

Hadoop-Based RDF Systems

SHARD (Rohloff and Schantz 2010) is one of the first Hadoop-based approaches that used a clause-iteration technique to evaluate SPARQL queries against RDF datasets. It is designed to exploit the MapReduce-style jobs for high parallelization of SPARQL queries. In particular, SHARD iterates over clauses in queries to incrementally bind query variables to the RDF graph nodes while conforming to all of the query constraints. In addition, an iteration algorithm is used to coordinate the iterated MapReduce jobs with one iteration for each clause in the query. In principle, the *Map* step filters each of the bindings, and the *Reduce* step removes duplicates where the key values for both *Map* and *Reduce* are the bound variables in the *SELECT* clause. The initial map step identifies all feasible bindings of graph data to variables in the first query clause. The output key of the initial map step is the list of variable bindings, and the output values are set to null. The intermediate MapReduce jobs continue to construct query responses by iteratively binding graph data to variables in later clauses as new

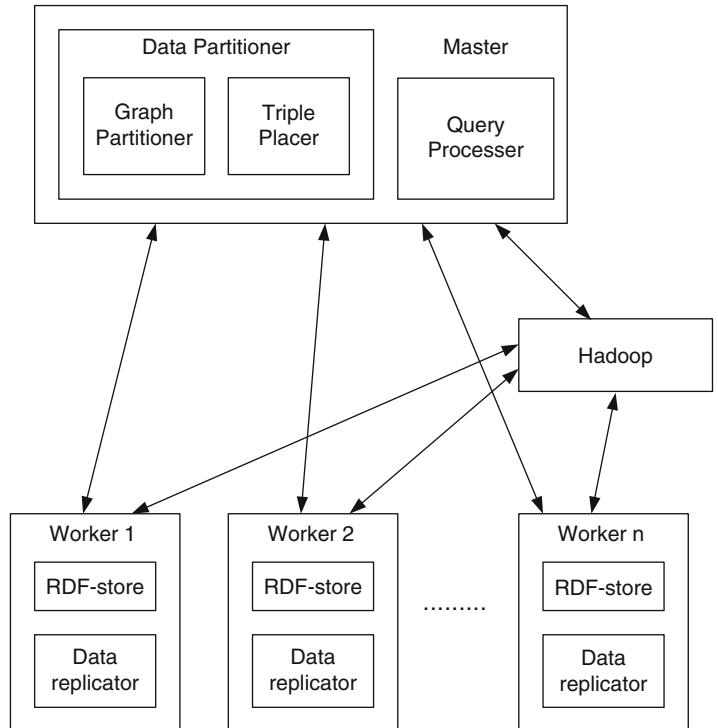
variables are introduced and then joining these new bindings to the previous bound variables such that the joined bound variables align with iteratively increasing subsets of the query clauses. The intermediate steps execute MapReduce operations simultaneously over both the graph data and the previously bound variables which were saved to disk to perform this operation. This iteration of MapReduce join continues until all clauses are processed and variables are assigned which satisfy the query clauses. The initial reduce step removes duplicate bindings without further modifying the output of the initial map step. A final MapReduce step consists of filtering bound variable assignments to obtain just the variable bindings requested in the *SELECT* clause of the original SPARQL query.

Husain et al. (2011) describe an approach to store RDF data in Hadoop Distributed File System (HDFS) via partitioning and organizing the data files and executing dictionary encoding. In particular, this approach applies two partitioning components. The *Predicate Split (PS)* component which splits the RDF data into predicate files. These predicate files are then fed into the *Predicate Object Split (POS)* component which splits the predicate files into smaller files based on the type of objects. Query processing is implemented on top of Hadoop by using a heuristic cost-based algorithm that produces a query plan with the minimal number of Hadoop jobs for joining the data files and evaluating the input query. The cost of the generated query plan is bounded by the log of the total number of variables in the given SPARQL query.

Figure 1 illustrates the architecture of the HadoopRDF system (Huang et al. 2011a) that has been introduced as a scale-out architecture which combines the distributed Hadoop framework with a centralized RDF store, **RDF-3X** (Neumann and Weikum 2010), for querying RDF databases. The data partitioner of HadoopRDF executes a disjoint partitioning of the input RDF graph by a vertex using a graph partitioning algorithm that allows triples which are close to each other in the RDF graph to be allocated on the same node. The main advantage of this partitioning strategy

**Framework-Based
Scale-Out RDF Systems,
Fig. 1**

MapReduce + RDF-3X:
system architecture (Huang
et al. 2011b). (©2011
VLDB Endowment.
Reprinted with permission)



is that it effectively reduces the network communication at query time. To further reduce the communication overhead, HadoopRDF replicates some triples on multiple machines. In particular, on each worker, the data replicator decides which triples are on the boundary of its partition and replicates them based on specified n-hop guarantees. HadoopRDF automatically decomposes the input query into chunks which can be evaluated independently with zero communication across partitions and uses the Hadoop framework to combine the resulting distributed chunks. It leverages HadoopDB (Abouzeid et al. 2009) to manage the partitioning of query evaluation across high-performance single-node database systems and the Hadoop framework.

The PigSPARQL system (Schätzle et al. 2013) compiles SPARQL queries into the Pig query language (Olston et al. 2008), a data analysis platform over the Hadoop framework. Pig uses a fully nested data model and provides relational style operators (e.g., filters and joins). In PigSPARQL, a SPARQL query is parsed

to generate an abstract syntax tree which is subsequently compiled into a SPARQL algebra tree. Using this tree, PigSPARQL applies various optimizations on the algebra level such as the early evaluation of filters and uses the selectivity information for reordering the triple patterns. Finally, PigSPARQL traverses the optimized algebra tree bottom-up and generates an equivalent sequence of Pig Latin expressions for every SPARQL algebra operator. For query execution, Pig automatically maps the resulting Pig Latin script onto a sequence of Hadoop jobs. An advantage of taking PigSPARQL as an intermediate layer that uses Pig between SPARQL and Hadoop is being independent of the actual Hadoop version or implementation details. RAPID+ (Ravindra et al. 2011) is another Pig-based system that uses an algebraic approach for optimizing and evaluating SPARQL queries on top of the Hadoop framework. It uses a data model and algebra (the Nested TripleGroup Data Model and Algebra (NTGA) Kim et al. 2013) which includes support for expressing graph pattern matching queries.

The **SHAPE** system (Lee and Liu 2013) uses a semantic hash partitioning approach that combines locality-optimized RDF graph partitioning with cost-aware query partitioning for processing queries over big RDF graphs. In practice, the approach utilizes access locality to partition big RDF graphs across multiple compute nodes by maximizing the intra-partition processing capability and minimizing the inter-partition communication cost. The **SHAPE** system is implemented on top of the Hadoop framework with the master server as the coordinator and the set of slave servers as the workers. In particular, RDF triples are fetched into the data partitioning module hosted on the master server, which partitions the data across the set of slave servers. The **SHAPE** system uses **RDF-3X** (Neumann and Weikum 2010) on each slave server and used Hadoop to join the intermediate results generated by subqueries. For query processing, the master node serves as the interface for SPARQL queries and performs distributed query execution planning for each query received. The **SHAPE** system classifies the query processing into two types: intra-partition processing and inter-partition processing. The intra-partition processing is used for the queries that can be fully executed in parallel on each server by locally searching the subgraphs matching the triple patterns of the query without any inter-partition coordination. The inter-partition processing is used for the queries that cannot be executed on any partition server, of which it needs to be decomposed into a set of subqueries such that each subquery can be evaluated by intra-partition processing. In this scenario, the processing of the query would require multiple rounds of coordination and data transfer across a set of partition servers. Thus, the main goal of the cost-aware query partitioning phase is to generate locality-optimized query execution plans that can effectively minimize the inter-partition communication cost for distributed query processing.

CliqueSquare (Goasdoué et al. 2015; Djahandideh et al. 2015) is another Hadoop-based RDF data management platform for storing and processing big RDF datasets. With the central

goal of minimizing the number of MapReduce jobs and the data transfer between nodes during query evaluation, CliqueSquare exploits the built-in data replication mechanism of the Hadoop Distributed File System (HDFS). Each of its partition has three replicas by default, to partition the RDF dataset in different ways. In particular, for the first replica, CliqueSquare partitions triples based on their subject, property, and object values. For the second replica, CliqueSquare stores all subject, property, and object partitions of the same value within the same node. Finally, for the third replica, CliqueSquare groups all the subject partitions within a node by the value of the property in their triples. Similarly, it groups all object partitions based on their property values. In addition, CliqueSquare implements a special treatment for triples whose property is *rdf:type*, by translating them into an unwieldy large property partition. CliqueSquare then splits the property partition of *rdf:type* into several smaller partitions according to their object value. For SPARQL query processing, CliqueSquare relies on a clique-based algorithm which produces query plans that minimize the number of MapReduce stages. The algorithm is based on the variable graph of a query and its decomposition into clique subgraphs. The algorithm works in an iterative way to identify cliques and to collapse them by evaluating the joins on the common variables of each clique. The process ends when the variable graph consists of only one node. Since triples related to a particular resource are colocated on one node, CliqueSquare can perform all first-level joins in RDF queries (SS, SP, SO, PP, PS, PO, etc.) locally on each node and reduce the data transfer through the network. In particular, it allows queries composed of 1-hop graph patterns to be processed in a single MapReduce job, which enables a significant performance competitive advantage.

Spark-Based RDF Systems

S2RDF (SPARQL on Spark for *RDF*) (Schätzle et al. 2015b) introduced a relational partitioning schema for encoding RDF data called ExtVP (the *Extended Vertical Partitioning*) that extends the vertical partitioning (VP) schema introduced by Abadi et al. (2007) and uses a semi-join-based

preprocessing to efficiently minimize query input size by taking into account the possible join correlations between the underlying encoding tables of the RDF data and join indices (Valduriez 1987). The main idea of the vertical partitioning scheme is to represent the RDF triple table into m tables where m is the number of unique properties in the RDF database. Each of the m table consists of two columns. The first column stores the subjects which are described by that property, while the second column stores the object values. The subjects which are not described by a particular property are simply omitted from the table for that property. ExtVP precomputes the possible join relations between partitions (i.e., tables). The main goal of ExtVP is to reduce the unnecessary I/O operations, comparisons, and memory consumption during executing join operations by avoiding the dangling tuples in the input tables of the join operations, i.e., tuples that do not find a join partner. In particular, S2RDF determines the subsets of a VP table VP_{p1} that are guaranteed to find at least one match when joined with another VP table VP_{p2} where $p1$ and $p2$ are query predicates. S2RDF uses this information to precompute a number of semi-join reductions (Bernstein and Chiu 1981) of VP_{p1} . The relevant semi-joins between tables in VP are determined by the possible joins that can occur when combining the results of triple patterns during query execution. Clearly, ExtVP comes at the cost of some additional storage overhead in comparison to the basic vertical partitioning techniques. Therefore, ExtVP does not use exhaustive precomputations for all the possible join operations. Instead, an optional selectivity threshold for ExtVP can be specified to materialize only the tables where reduction of the original tables is large enough. This mechanism facilitates the ability to control and reduce the size overhead while preserving most of its performance benefit. S2RDF uses the Parquet columnar storage format for storing the RDF data on the Hadoop Distributed File System (HDFS). S2RDF is built on top of Spark. The query evaluation of S2RDF is based on SparkSQL (Armbrust et al. 2015), the relational interface of Spark. It parses a SPARQL query into

the corresponding algebra tree and applies some basic algebraic optimizations (e.g., filter pushing) and traverses the algebraic tree bottom-up to generate the equivalent Spark SQL expressions. For the generated SQL expression, S2RDF can use the precomputed semi-join tables, if they exist, or alternatively uses the base encoding tables.

SparkRDF (Chen et al. 2014, 2015) is another Spark-based RDF engine which partitions the RDF graph into MESGs (Multi-layer Elastic SubGraphs) according to relations (R) and classes (C) by building five kinds of indices (C, R, CR, RC, CRC) with different granularities to support efficient evaluation for the different query triple patterns. SparkRDF creates an index file for every index structure and stores such files directly in the HDFS, which are the only representation of triples used for the query execution. These indices are modeled as RDGSs (Resilient Discreted SubGraphs), a collection of in-memory subgraphs partitioned across nodes. SPARQL queries are evaluated over these indices using a series of basic operators (e.g., filter, join). All intermediate results are represented as RDGSs and maintained in the distributed memory to support faster join operations. SparkRDF uses a selectivity-based greedy algorithm to build a query plan with an optimal execution order of query triple patterns that aims to effectively reduce the size of the intermediate results. In addition, it uses a location-free pre-partitioning strategy that avoids the expensive shuffling cost for the distributed join operations. In particular, it ignores the partitioning information of the indices while repartitioning the data with the same join key to the same node.

The **S2X** (SPARQL on Spark with GraphX) (Schätzle et al. 2015a) RDF engine has been implemented on top of GraphX (Gonzalez et al. 2014), an abstraction for graph-parallel computation that has been augmented to Spark (Zaharia et al. 2010). It combines graph-parallel abstractions of GraphX to implement the graph pattern matching constructs of SPARQL. A similar approach has been followed by Goodman and Grunwald (Goodman and Grunwald 2014) for implementing an RDF engine on top the

GraphLab framework, another graph-parallel computation platform (Low et al. 2012). Naacke et al. (2016) compared five Spark-based SPARQL query processing based on different join execution models. The results showed that hybrid query plans combining partitioned join and broadcast joins improve query performance in almost all cases.

Cross-References

- ▶ [Federated RDF Query Processing](#)
- ▶ [Hadoop](#)
- ▶ [Native Distributed RDF Systems](#)

References

- Abadi DJ, Marcus A, Madden SR, Hollenbach K (2007) Scalable semantic web data management using vertical partitioning. In: Proceedings of the 33rd international conference on very large data bases. VLDB Endowment, pp 411–422
- Abouzeid A, Bajda-Pawlikowski K, Abadi DJ, Rasin A, Silberschatz A (2009) Hadoopdb: an architectural hybrid of mapreduce and DBMS technologies for analytical workloads. PVLDB 2(1):922–933. <http://www.vldb.org/pvldb/2/vldb09-861.pdf>
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: SIGMOD. <https://doi.org/10.1145/2723372.2742797>
- Bernstein PA, Chiu DMW (1981) Using semi-joins to solve relational queries. J ACM (JACM) 28(1): 25–40
- Chen X, Chen H, Zhang N, Zhang S (2014) SparkRDF: elastic discretized RDF graph processing engine with distributed memory. In: Proceedings of the ISWC 2014 posters & demonstrations track a track within the 13th international semantic web conference, ISWC 2014, Riva del Garda, 21 Oct 2014, pp 261–264. http://ceurws.org/Vol-1272/paper_43.pdf
- Chen X, Chen H, Zhang N, Zhang S (2015) SparkRDF: elastic discretized RDF graph processing engine with distributed memory. In: IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology, WI-IAT 2015, Singapore, 6–9 Dec 2015, vol I, pp 292–300. <https://doi.org/10.1109/WI-IAT.2015.186>
- Dean J, Ghemawa S (2004) MapReduce: simplified data processing on large clusters. In: OSDI
- Djahandideh B, Goasdoué F, Kaoudi Z, Manolescu I, Quiané-Ruiz J, Zampetakis S (2015) Cliquesquare in action: flat plans for massively parallel RDF queries. In: 31st IEEE international conference on data engineering, ICDE 2015, Seoul, 13–17 Apr 2015, pp 1432–1435. <https://doi.org/10.1109/ICDE.2015.7113394>
- Goasdoué F, Kaoudi Z, Manolescu I, Quiané-Ruiz J, Zampetakis S (2015) Cliquesquare: flat plans for massively parallel RDF queries. In: 31st IEEE international conference on data engineering, ICDE 2015, Seoul, 13–17 Apr 2015, pp 771–782. <https://doi.org/10.1109/ICDE.2015.7113332>
- Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) GraphX: graph processing in a distributed dataflow framework. In: OSDI. <https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez>
- Goodman EL, Grunwald D (2014) Using vertex-centric programming platforms to implement SPARQL queries on large graphs. In: Proceedings of the 4th workshop on irregular applications: architectures and algorithms, IA3 '14. IEEE Press, Piscataway, pp 25–32. <https://doi.org/10.1109/IA3.2014.10>
- Huang J, Abadi DJ, Ren K (2011a) Scalable SPARQL querying of large RDF graphs. PVLDB 4(11): 1123–1134
- Huang J, Abadi DJ, Ren K (2011b) Scalable SPARQL querying of large RDF graphs. Proc VLDB Endow 4(11):1123–1134
- Husain M, McGlothlin J, Masud MM, Khan L, Thuraisingham BM (2011) Heuristics-based query processing for large RDF graphs using cloud computing. IEEE Trans Knowl Data Eng 23(9): 1312–1327
- Kim H, Ravindra P, Anyanwu K (2013) Optimizing RDF(S) queries on cloud platforms. In: 22nd international world wide web conference, WWW '13, Rio de Janeiro, 13–17 May 2013, Companion volume, pp 261–264. <http://dl.acm.org/citation.cfm?id=2487917>
- Lee K, Liu L (2013) Scaling queries over big RDF graphs with semantic hash partitioning. Proc VLDB Endow 6(14):1894–1905
- Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM (2012) Distributed graphLab: a framework for machine learning in the cloud. PVLDB 5(8):716–727
- Naacke H, Curé O, Amann B (2016) SPARQL query processing with Apache spark. CoRR abs/1604.08903. <http://arxiv.org/abs/1604.08903>
- Neumann T, Weikum G (2010) The RDF-3x engine for scalable management of RDF data. VLDB J 19(1): 91–113
- Olston C, Reed B, Srivastava U, Kumar R, Tomkins A (2008) Pig latin: a not-so-foreign language for data processing. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2008, Vancouver, 10–12 June 2008, pp 1099–1110. <https://doi.org/10.1145/1376616.1376726>

- Poggi A, Lembo D, Calvanese D, De Giacomo G, Lenzerini M, Rosati R (2008) Linking data to ontologies. In: Spaccapietra S (ed) *Journal on data semantics X*. Springer, Berlin/Heidelberg, pp 133–173
- Ravindra P, Kim H, Anyanwu K (2011) An intermediate algebra for optimizing RDF graph pattern matching on mapreduce. In: The semantic web: research and applications – 8th extended semantic web conference, ESWC 2011, Heraklion, Crete, 29 May – 2 June 2011, Proceedings, Part II, pp 46–61. https://doi.org/10.1007/978-3-642-21064-8_4
- Rohloff K, Schantz RE (2010) High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store. In: Programming support innovations for emerging distributed applications. ACM, p 4
- Sakr S (2016) Big data 2.0 processing systems – a survey. Springer briefs in computer science. Springer. <https://doi.org/10.1007/978-3-319-38776-5>
- Sakr S, Liu A, Fayoumi AG (2013) The family of mapreduce and large-scale data processing systems. ACM Comput Surv 46(1). <https://doi.org/10.1145/2522968.2522979>
- Schätzle A, Przyjaciel-Zablocki M, Hornung T, Lausen G (2013) Pigsparql: a SPARQL query processing baseline for big data. In: Proceedings of the ISWC 2013 posters & demonstrations track, Sydney, 23 Oct 2013, pp 241–244. http://ceur-ws.org/Vol-1035/iswc2013_poster_16.pdf
- Schätzle A, Przyjaciel-Zablocki M, Berberich T, Lausen G (2015a) S2X: graph-parallel querying of RDF with GraphX. In: 1st international workshop on big-graphs online querying (Big-O(Q))
- Schätzle A, Przyjaciel-Zablocki M, Skilevic S, Lausen G (2015b) S2RDF: RDF querying with SPARQL on spark. CoRR abs/1512.07021. <http://arxiv.org/abs/1512.07021>
- Valduriez P (1987) Join indices. ACM Trans Database Syst 12(2):218–246. <https://doi.org/10.1145/22952.22955>
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: HotCloud

Functional Benchmark

► Microbenchmark

G

Gauge

► Metrics for Big Data Benchmarks

Genomic Data Compression

Kaiyuan Zhu¹, Ibrahim Numanagić², and S. Cenk Sahinalp¹

¹Department of Computer Science, Indiana University Bloomington, Bloomington, IN, USA

²Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Definitions

High-throughput sequencing platforms generate millions of short reads from a genome sequence, stored in a specific (FASTQ/SAM) format along with other information. Since each position in the genome is redundantly covered by many different reads, genomic data compression algorithms aim to reduce the burden in (high-throughput sequencing) data storage and transmission by reducing this redundancy.

Overview

Current trends in HTS data generation indicate that storage, transmission, and bandwidth costs will soon surpass the costs of sequencing and

become the main bottleneck in genomics as well as in the applications of HTS data to precision medicine. Most HTS data are maintained either as raw sequencing information in a FASTQ file or as reference-aligned (mapped) data in Sequence Alignment/Map (SAM) format. Although they have fundamental differences, the basic FASTQ and SAM schemata are both comprised of distinct data fields with specific properties; e.g., the sequence field consists only of nucleotide reads with length in the order of a few hundred to few thousand base pairs, while the mapping loci (on the reference) are usually represented as a nondecreasing sequence of integers. Much of existing HTS data is compressed by general-purpose compression tools, which treat them as simple plaintext files and thus produce suboptimal compression rates because they are not able to exploit the underlying data schemata.

Specialized FASTQ compressors initially perform a form of data transformation followed by statistical modeling and entropy coding. More advanced approaches achieve significantly higher compression ratios on FASTQ files by replacing each read with a pointer to the underlying reference genome or by reordering the FASTQ records in a manner that brings similar reads together, since the order of the records in FASTQ format is arbitrary. SAM files are mostly stored in their compressed binary equivalent, the BAM format. An alternative to the SAM format is CRAM, a reference-based format that separates different fields in the reads and applies a variety of compression techniques on them. The best

performing SAM compressors encode reads carrying the same sequence variants only once or employ highly optimized statistical models for various SAM fields.

In a recent benchmarking study to evaluate the effectiveness of available compression methods, the MPEG HTS compression working group compiled an HTS data set consisting of both raw (FASTQ) and aligned (SAM or BAM) read data with a wide spectrum of characteristics for ensuring statistically meaningful results (Numanagić et al. 2016). On this data set, compression performance, running times, memory usage, and parallelization capabilities were measured for available compression tools, including both industry-scale tools and research-oriented prototypes. The study declared no overall winner that can perform well on each data type and under every performance measure employed. Therefore, an integrated solution that chooses the specific approach offering the best performance for the specific application and the specific input data type(s) (with respect to the performance measure of importance) would yield the best outcome, both for raw and aligned sequence data.

FASTQ Compression Tools

FASTQ is a standard text format for storing unmapped HTS data provided as a set of reads (Cock et al. 2009). Each record (representing a read) in a FASTQ file consists of read identifier, sequence information, comment, and quality score (Ewing et al. 1998) information, each separated by newline characters. If the sequencer is able to produce *paired-end* reads, FASTQ files usually come in the so-called *library* format consisting of two FASTQ files, each consisting of the reads from the associated strand, such that the k -th read from first file corresponds to the k -th read from the second file.

General FASTQ Compressors

FASTQ files are many times compressed with general-purpose tools, such as Gzip (Deutsch 1996) and bzip2 (Seward 1998), resulting in

suboptimal compression ratios since they treat FASTQ as a simple plaintext file. Another popular format, although only internally used in the large sequencing databanks, is NCBI's SRA (Leinonen et al. 2010), which uses the LZ-77 scheme to encode metadata.

General FASTQ compressors typically decouple distinct streams (read names, sequences, comments, and quality scores) and compress each of them separately. Examples include DSRC and DSRC2 (Deorowicz and Grabowski 2011; Roguski and Deorowicz 2014), FQC (Dutta et al. 2015), LFQC (Nicolae et al. 2015), Fqzcomp (Bonfield and Mahoney 2013; Holland and Lynch 2013), and Slimfastq (Josef 2014). Instead of directly compressing each stream as text symbols, these tools may first perform reformatting on each field, e.g., read identifier *tokenization* to encode the common prefix of read identifiers only once or 2-bit nucleotide encoding. Quality scores are usually compressed by statistical modeling and entropy coding.

Alignment-Based FASTQ Compressors

The order of records within a FASTQ file is arbitrary, and thus an alteration of their order for the purpose of improving compression performance would not result in loss of information. Indeed, some of the newer approaches reorder reads in an input FASTQ file, typically with the goal of approximating their order within the genome sequence. Such reordering is helpful, e.g., when coupled with an encoding scheme that replaces each read with a pointer to a reference genome (together with a list of mismatches, if the alignment is not perfect). The reference can either be user-provided or constructed via *de novo assembly*, which is usually performed by representing the read collection as a de Bruijn graph and traversing it in a way to “cover” all reads.

Provided with a reference, alignment-based compression tools (explicitly or implicitly) sort the reads with respect to their mapping position in the reference. The reads, represented as pointers to the reference, could then be encoded, e.g., via run-length encoding. Naturally the compression performance of this general approach increases

with the read *coverage* (average number of reads that “cover” each position in the genome) and the repeat content of the genome.

Tools that employ this general strategy include LW-FQZip (Zhang et al. 2015) which uses an available reference genome for extracting the pointers, as well as Quip (Jones et al. 2012), Leon (Benoit et al. 2015), k-Path (Kingsford and Patro 2015), and KIC (Zhang et al. 2016) which first perform a de novo assembly of reads into longer contigs.

Reordering-Based FASTQ Compressors

Although the alignment-based methods mentioned above achieve good compression performance, they are typically slow since explicit read mapping and de novo sequence assembly are computation-intensive tasks. Perhaps the best trade-off between the compression rate and running time is achieved by compression methods that perform read mapping or assembly implicitly. Such methods first cluster reads that share long substrings (typically in an online fashion) and then independently compress reads in each cluster after reordering them or assembling them into longer contigs.

The first reordering-based FASTQ compression method, SCALCE (Hach et al. 2012), clusters reads based on their *core substrings*, a small subset of substrings from the input alphabet (in this case, the *four-letter DNA alphabet*) of a given length range, that are guaranteed to be present in all reads. For each read, SCALCE uses the lexicographically smallest core substring in the read for clustering purposes. Follow-up tools, such as Orcom (Grabowski et al. 2014) and Mince (Patro and Kingsford 2015), cluster the input reads into distinct sets according to a variety of *minimizer* (Roberts et al. 2004) schemes (similar to the core substrings, a minimizer is defined as the lexicographically smallest *k*-mer within the read that satisfies certain properties). Another tool, BEETL, uses a generalized version of BWT that operates on sets of strings for reordering (rather than clustering) reads (Cox et al. 2012); somewhat similarly, a more recent tool, HARC, uses a hash-based reordering of reads for improved performance (Chandak et al.

2018). Finally Assembltrie uses a fast, greedy assembly of the reads into trees (Ginart et al. 2018) (rather than strings) to achieve combinatorial optimality with respect to the number of symbols used in the reference constructed. Note that some of the abovementioned compression tools are research-oriented products and do not support compression of read identifiers/quality scores.

G

SAM Compression Tools

The SAM format Sam/bam Format Specification Working Group (2014) is a tab-delimited plaintext representation of sequences mapping to an arbitrary reference genome. It includes all the data from FASTQ files, accompanied by additional mapping information for every read, including (but not limited to) the position within the reference genome and the edit operations (so-called CIGAR strings) necessary to derive the reads from a reference. SAM files are many times stored in their binary compressed equivalent, the BAM format, which is a variation of LZ-77 scheme that allows random access in the compressed file with a negligible overhead. In addition to Samtools (Li et al. 2009), the most widely used software tool for SAM/BAM processing, a few other alternatives also use BAM as their output format, such as Picard Tools – By Broad Institute (2015) and Sambamba (Tarasov et al. 2015).

The CRAM Format and Reference-Based Compressors

As an alternative to SAM/BAM (that does not require any reference during the compression or decompression process), the reference-based CRAM CRAM format specification (version 3.0) format offers improved compression by handling distinct data fields separately. CRAM is implemented in Cramtools (Fritz et al. 2011), Scramble (Bonfield 2014), and more recently also in Samtools and Picard. In particular, Scramble can also compress reads without any reference, at the cost of lower compression rate; many of the

improvements offered by Scramble are now part of the CRAM format.

Variation-Sensitive Compression and Other SAM Compressors

In both SAM and CRAM files, reads harboring the same sequence variation are redundantly encoded due to independent encoding of each read. In order to eliminate this redundancy, a newer tool, DeeZ (Hach et al. 2014) implicitly assembles the underlying genome in order to encode these variations only once. A similar approach is followed by CBC (Ochoa et al. 2014) and TSC (Voges et al. 2016). All of these tools treat distinct SAM fields separately and apply a variety of compression techniques on each field. Finally, some of the best-performing tools in terms of compression rate, such as Quip and sam_comp (Bonfield and Mahoney 2013), employ highly optimized statistical models for various SAM fields at the expense of slower decompression.

Final Remarks

A recent benchmarking study on existing HTS compression methods (Numanagić et al. 2016) provided a comprehensive comparison of FASTQ and SAM compressors, revealing that some of the newly available tools achieve significant gains over universal lossless compression methods such as Gzip and bzip2 (as well as the standard Samtools for SAM compression) both in terms of compression rate and speed. Some of these tools provide significantly better compression rates at the expense of higher running time (both for compression and decompression) or larger memory usage. It is also important to note that the majority of the available tools are optimized for Illumina-style short, fixed-length reads (typically only a few hundred base pairs long) with a low ($\sim 1\%$) sequencing error rate. Compressing long, variable-length and error-prone read collections, such as data obtained with sequencers from

Pacific Biosciences or Oxford Nanopore, has proven to be a more challenging task.

References

- Benoit G et al (2015) Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. *BMC Bioinformatics* 16:288
- Bonfield JK (2014) The scramble conversion tool. *Bioinformatics* 30(19):2818–2819
- Bonfield JK, Mahoney MV (2013) Compression of FASTQ and SAM format sequencing data. *PLoS one* 8:e59190
- Chandak S, Tatwawadi K, Weissman T (2018) Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis. *Bioinformatics* 34:558–567
- Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM (2009) The sanger FASTQ file format for sequences with quality scores, and the solexa/illumina FASTQ variants. *Nucleic Acids Res* 38:1767–1771
- Cox AJ, Bauer MJ, Jakobi T, Rosone G (2012) Large-scale compression of genomic sequence databases with the burrows-wheeler transform. *Bioinformatics* 28:1415–1419
- CRAM format specification (version 3.0) (2017) <https://samtools.github.io/hts-specs/CRAMv3.pdf>
- Deorowicz S, Grabowski S (2011) Compression of DNA sequence reads in FASTQ format. *Bioinformatics* 27:860–862
- Deutsch LP (1996) GZIP file format specification version 4.3. <https://tools.ietf.org/html/rfc1952>
- Dutta A, Haque MM, Bose T, Reddy CVSK, Mande SS (2015) FQC: a novel approach for efficient compression, archival, and dissemination of fastq datasets. *J Bioinform Comput Biol* 13:1541003
- Ewing B, Hillier L, Wendl MC, Green P (1998) Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Res* 8:175–185
- Fritz MHY, Leinonen R, Cochrane G, Birney E (2011) Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res* 21:734–740
- Ginart AA et al (2018) Optimal compressed representation of high throughput sequence data via light assembly. *Nat Commun* 9:566
- Grabowski S, Deorowicz S, Roguski Ł (2014) Disk-based compression of data from genome sequencing. *Bioinformatics* 31:1389–1395
- Hach F, Numanagić I, Alkan C, Sahinalp SC (2012) SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics* 28:3051–3057
- Hach F, Numanagić I, Sahinalp SC (2014) DeeZ: reference-based compression by local assembly. *Nat Methods* 11:1082–1084
- Holland RC, Lynch N (2013) Sequence squeeze: an open contest for sequence compression. *GigaScience* 2:5

- Jones DC, Ruzzo WL, Peng X, Katze MG (2012) Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Res* 40:e171–e171
- Josef E (2014) Fast, efficient, lossless compression of FASTQ files. <https://github.com/Infinidat/slimfastq>
- Kingsford C, Patro R (2015) Reference-based compression of short-read sequences using path encoding. *Bioinformatics* 31:1920–1928
- Leinonen R, Sugawara H, Shumway M, International Nucleotide Sequence Database Collaboration (2010) The sequence read archive. *Nucleic Acids Res* 39:D19–D21
- Li H et al (2009) The sequence alignment/map format and SAMtools. *Bioinformatics* 25:2078–2079
- Nicolae M, Pathak S, Rajasekaran S (2015) LFQC: a lossless compression algorithm for FASTQ files. *Bioinformatics* 31:3276–3281
- Numanagić I et al (2016) Comparison of high-throughput sequencing data compression tools. *Nat Methods* 13:1005–1008
- Ochoa I, Hernaez M, Weissman T (2014) Aligned genomic data compression via improved modeling. *J Bioinform Comput Biol* 12:1442002
- Patro R, Kingsford C (2015) Data-dependent bucketing improves reference-free compression of sequencing reads. *Bioinformatics* 31:2770–2777
- Picard Tools – By Broad Institute (2015) <http://broadinstitute.github.io/picard/>
- Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA (2004) Reducing storage requirements for biological sequence comparison. *Bioinformatics* 20:3363–3369
- Roguski Ł, Deorowicz S (2014) DSRC 2industry-oriented compression of FASTQ files. *Bioinformatics* 30: 2213–2215
- Sam/bam Format Specification Working Group et al (2014) Sequence alignment/map format specification. <http://samtools.github.io/hts-specs/SAMv1.pdf>
- Seward J (1998) bzip2. <http://www.bzip.org/>
- Tarasov A, Vilella AJ, Cuppen E, Nijman IJ, Prins P (2015) Sambamba: fast processing of NGS alignment formats. *Bioinformatics* 31:2032–2034
- Voges J, Munderloh M, Ostermann J (2016) Predictive coding of aligned next-generation sequencing data. In: Data compression conference (DCC 2016). IEEE, pp 241–250
- Zhang Y et al (2015) Light-weight reference-based compression of FASTQ data. *BMC Bioinformatics* 16:188
- Zhang Y, Patel K, Endrawis T, Bowers A, Sun Y (2016) A FASTQ compressor based on integer-mapped k-mer indexing for biologist. *Gene* 579:75–81

Geo-distributed Transaction Processing

- Geo-scale Transaction Processing

Geo-replicated Storage

- Geo-replication Models

G

Geo-replicated Transaction Processing

- Geo-scale Transaction Processing

Geo-replicated Transactions

- Achieving Low Latency Transactions for Geo-replicated Storage with Blotter

Geo-replication Models

Mahsa Najafzadeh and Suresh Jagannathan
Purdue University, West Lafayette, IN, USA

Synonyms

Data storage replication; Geo-replicated storage; Multi-data center replication

Definitions

Geo-replication copies data in multiple data centers or sites across the globe, in order to bring data closer to the clients. Users can access a nearby data center, and perform operations locally, if possible. This local access improves latency by avoiding the high cost of network latency between data centers and increases availability in the presence of failures (Corbett et al. 2012; DeCandia et al. 2007; Shapiro et al. 2011; Lloyd et al. 2011). For instance, the data for a bank account can be replicated at the bank's branches

in different locations around the world. A bank user can access a local branch and perform transactions.

Overview

Geo-replication models have emerged as an important technique for distributed applications over the Internet, such as cloud computing. Wide area network (WAN) communication becomes a fundamental barrier for developing the large-scale Internet-based applications. User requests incur at least one round-trip across a WAN. While this might be acceptable for some applications, such as web browsing, an important class of applications (e.g., online game applications and e-commerce applications) is actually latency sensitive. For instance, Amazon reported that every 100ms slowdown degrades revenue by one percent. The WAN latency issue becomes more challenging as the market for distributed systems is being fueled by cloud computing, big data, exascale computing, and so on.

Geo-replicated systems decrease WAN-based latency by bringing data closer to users. The replicated system model consists of a set of data centers, each maintaining a copy of the database state. A database contains a set of replicated data items. Each data item has a *value* at any one time and a *type* that determines the set of operations that can be invoked on the data object. Users interact with the database system through running applications. A client accesses a single replica, referred as its *local* replica, and invokes a set of operations on the data objects stored in the database.

Different geo-replicated systems offer different services, ranging from simple *read* and *write* operations into ACID transactions (Gray and Reuter 1993). COPS (Lloyd et al. 2011) and Amazon's Dynamo (DeCandia et al. 2007) are examples of a key-value geo-replicated storage system with read and write operations. COPS (Lloyd et al. 2011) supports read-only operations with a consistent view of multiple keys, but it does not allow multi-key write operations. Eiger (Lloyd et al. 2013), the extended version

of COPS, supports both read-only and write-only transactions across multiple keys. However, Eiger does not support arbitrary transactions with reads and writes across multiple keys.

Some geo-replicated systems, such as Cassandra (Lakshman and Malik 2010) and Yahoo's PNUTS (Cooper et al. 2008), extend read and write semantics by allowing clients to perform high-level update and query operations over data objects. A client's query simply returns the value of an object without changing the database state, but update operations modify the state. Consider PNUTS (Cooper et al. 2008), a geo-replicated data storage that supports a simple relational data model, organizing data into tables of records (or objects). Each record has a version; updating a record creates a new version for the record. PNUTS's API provides several primitive calls for read and update requests with different guarantees. For instance, PNUTS's *read-only* semantics allows a local replica to return a stale version of data, while PNUTS's *read-latest* semantics returns the most current version of data. For update requests, PNUTS allows to atomically create or update records and its update API includes a test-and-set semantics, which allows updating a record only if current version of that record is equals to a given required version.

Recent geo-replicated systems support arbitrary transactions among multiple data objects, which enable the encapsulation of a group of operations into a single unit, guaranteeing certain pleasant properties, i.e., atomicity, consistency, isolation, and durability (ACID). For instance, Google's Spanner (Corbett et al. 2012) proposes a transactional data storage with ACID semantics across multiple data centers.

To maintain state of all replicas of a geo-replicated model consistently, there are two main synchrony protocols that are used: *synchronous replication* and *asynchronous replication*.

Synchronous Replication

Ideally, the geo-replicated system must ensure *strong consistency*, providing the behavior of a single centralized system. Implementing strong consistency semantics, such as linearizability (Herlihy and Wing 1990) and serializability

(Papadimitriou 1979), requires a global total order of updates (*synchronous replication*). Linearizability provides the intuition of a single replica that executes all operations in a real-time order. Serializability (SER) ensures that every execution of operations is equivalent to some serial execution. In a synchronous replication approach, when a user issues a request at a local replica, before proceeding with the request, this replica must synchronize with all other replicas to find out if there are some concurrent conflicts, such as writing to the same data object. For instance, in a bank account application with deposit and debit operations, if one replica wants to perform a debit operation, it must first synchronize with all other replicas to stop other concurrent debits that would make the balance negative.

Such a synchronous protocol ensures that all replicas execute application's requests in the same order, and if some operations conflict, only one of these operations can be accepted; declined operations may be ignored or the application may choose to retry them. This synchronous and strongly consistent approach simplifies reasoning about the behavior of a program in a geo-replicated system, but the synchronization comes at a significant cost in availability (or performance).

Examples

Google's Megastore (Baker et al. 2011) implements a synchronous geo-replication model for a NoSQL data store where data is partitioned into small databases, called *entity groups*. Each entity group is synchronously replicated across data centers via Paxos consensus protocol (Gray and Lamport 2006). A replicated transaction log is assigned to each entity group. Megastore provides transactional ACID semantics within an entity group by enforcing transactions which have access the entity group mutually access the log's group, i.e., only one transaction can update the log at any time. Such synchronous approach serializes transactions running in the same group.

Google's Spanner (Corbett et al. 2012) is another geo-replicated model that supports a synchronous replication model across data centers

using Paxos (Gray and Lamport 2006), ordering transactions with regard to a real-time clock, i.e., transactions commit in an order equivalent to their real-time order. Consider in a bank application, Alice transfers 100 dollar into Bob's account, and sometime later Bob reads his account. A database providing real-time ordering will guarantee that Bob sees 100 dollar in his account. To implement the real-time ordering in a distributed system, Spanner introduces a new API, called *TrueTime*, which bounds the clock uncertainty by specifying an error margin for each local physical time clock. So, a transaction's timestamp is an interval between the earliest time that transaction could possibly be and the latest time. Given the interval, Spanner orders transaction T_1 before transaction T_2 iff $T_1.\text{latest} < T_2.\text{earliest}$. To ensure atomicity and isolation within a data center, Spanner relies on two phase commit and two phase locking.

Asynchronous Replication

Experience with real-world applications shows the synchronous approach often causes more synchronization than the application really needs (Bailis et al. 2013; Wang et al. 2002). To avoid this cost, one alternative approach is to allow a replica to execute a client's request without synchronizing a priori with another replicas. Instead, it returns a response immediately to the client; this request is propagated to all replica in background. Eventually, every replica performs the same request, possibly with a delay. We call this *asynchronous replication*.

For instance, COPS (Lloyd et al. 2011, 2013) is a key-value geo-replicated system that supports asynchronous writes; each client could access a copy of data stored in his local replica and independently perform operations.

Concurrency and Conflict Resolutions

Consistency problems arise when multiple replicas independently (i.e., without synchronizing with other replicas) update the same data object. Concurrent updates to multiple replicas of the same data object may execute in different orders among replicas; this may cause *conflicts*. For instance, consider Alice and Bob both access a

shared file *foo* at different replicas r_1 and r_2 , respectively. Type of file *foo* is registered with operations to read the value or to write a new value. A write to the register rewrites its last successful written value. Alice writes *a* to the file *foo*. Concurrently, Bob writes *b* to the same file. After exchanging the updates, content of file *foo* will be different at replicas r_1 and r_2 . This violates the expectation of convergence to the same state. A replicated system is said to be convergent if all replicas that observed the same set of updates have equivalent state.

There are different approaches used by replicated systems to resolve such conflicts raised by concurrent executions. A common policy is to accept one update of the two conflicting updates and ignore the other, for example, COPS (Lloyd et al. 2011) exploits *last-writer-win* rule(LWW) (also called Thomas's write rule (Johnson and Thomas 1976)) that assigns a unique timestamp to every update, and in case of concurrent conflicts, the update with a higher timestamp is selected and the older one is ignored.

The second approach is that the replicated system may support some application-specific conflict resolutions (DeCandia et al. 2007; Satyanarayanan et al. 1990; Terry et al. 1995). For example, the data store simply keeps the set of all concurrently written values of an object and presents them to the application. It is up to the application how to resolve the conflicts. This approach is prevalent in replicated file systems, such as LOCUS (Popek et al. 1981), Coda (Satyanarayanan et al. 1990), Ficus (Guy et al. 1990), and Roam (Ratner et al. 1999), and in version control systems, such as Git or SVN.

Alternatively, a replicated system may take advantage of replicated data types, such as CRDTs (Shapiro et al. 2011), which automatically resolve conflicting updates, ensuring that database itself has knowledge of resolution semantics in order to integrate the conflict resolution in the replication protocol.

Therefore, an asynchronous geo-replicated system may use one or more conflict resolution strategies to *optimistically* resolve conflicts. For instance, Walter (Sovran et al. 2011) is a geo-replicated transactional key-value data store that

uses two approaches: preferred replicas and counting set (cset). The idea of preferred copy is similar to primary copy in state machine replication model. Each object is assigned a preferred replica where updates into the object can be performed locally. A transaction commits locally within a single replica r if the preferred replica of all objects written by this transaction is r . Otherwise, committing the transaction in a non-preferred replica requires synchronization with all other replicas. Although the preferred replica approach amortizes the cost of distributed transactions, it does not scale well if data objects modified by multiple users in different replicas. Walter provides another technique built upon a commutative data type, called *cset* to avoid conflicts. Like commutative data types (Shapiro et al. 2011), cset supports commutative operations, which address write conflicts on the same data objects.

Application Invariants

An application might have a set of invariants, which are *explicitly* stated. An invariant is a property over the database state that determines the safe behavior of the application in terms of correct values that may be observed by users accessing the data objects. Unfortunately, asynchronous replication trades sequential safety semantics for availability. For instance, the bank application may have an invariant enforcing that the account balance be always non-negative. If the balance is initially \$100, and two users, connecting to different bank branches, try to withdraw \$60, without synchronization, both withdrawals will succeed but the balance will become negative(\$−20).

Key Research Findings

A major challenge in the design of geo-replicated systems is to ensure consistency among data centers. The CAP theorem (Gilbert and Lynch 2002) asserts a replicated system cannot promise both availability and consistency in the presence of failures. Thus, designers of geo-replicated system face a vexing choice between exploiting strong consistency guarantees of synchronous

replication, which promises standard sequential behavior to applications but is slow and fragile, and asynchronous replication, which is highly available and responsive but exposes users to concurrency anomalies.

No single replication model can best suit all purposes: it can provide acceptable consistency requirements for all applications without sacrificing performance and availability (Sovran et al. 2011).

Recent geo-replicated systems, such as Gemini (Li et al. 2012) and Walter Sovran et al. (2011), are evolving toward *hybrid* models, which provide different levels of consistency depending on the given application requirements. In the hybrid approach, the replicated data storage provides the asynchronous replication model by default, and synchronizations are added upon request (Li et al. 2012; Balegas et al. 2015; Terry et al. 2013). For instance, Gemini (Li et al. 2012) is a geo-replicated system implementing a hybrid consistency model, called *RedBlue*. RedBlue consistency model allows both synchronous and asynchronous replication models to coexist by classifying application operations as *red* and *blue*. Blue operations commute with all others; they are fast and always available even when disconnected. Red operations must be totally ordered, requiring system-wide synchronization. RedBlue relies on programmer-specified invariants to determine where synchronization is necessary. Consider a replicated bank application where clients can access different bank branches and perform deposit and debit operations. The application invariant is the account balance to be non-negative. Deposit operations are blue, i.e., clients can add to their account in all circumstances, but debits are red, because the system needs to stop a debit that would make the balance negative. If the client cannot connect with the bank server, all debits are blocked.

Examples of Application

Geo-replicated systems bring tremendous benefits for applications with high availability and

scalability requirements from social networks (Wittie et al. 2010) (e.g., Facebook and Twitter rely on geo-replicated storage) and multi-player online games to e-commerce applications such as Amazon (DeCandia et al. 2007).

Future Directions for Research

Navigating the trade-off between consistency and performance (or availability) is not clear-cut in a geo-replicated system. Although hybrid consistency models significantly improve performance, there are still lots of issues that must be addressed to exploit such techniques effectively; this requires carefully analyzing application semantics and identifying its consistency requirements. Static analysis and verification approaches are promising. They help programmers remove synchronization from critical executions path, adding synchronization only when strictly needed.

A number of verification tools for reasoning about application executions in a geo-replicated setting have been already proposed (Li et al. 2014; Gotsman et al. 2016; Kaki et al. 2018). Li et al. (2014) have presented a static analysis in order to classify operations into synchronous and asynchronous operations in the redblue consistency model. The analysis checks if executing operations preserves a given integrity invariant; if not, the analysis concludes that the operations require synchronization, i.e., it is a red operation. Bailis et al. (2015) have proposed a necessary and sufficient condition, called I-confluent analysis, to check whether operation executions on a replicated database need synchronization or not. The work analyzes various operations of an application and its desired invariants in order to detect necessary synchronization. Recently developed static analysis tools (Gotsman et al. 2016; Kaki et al. 2018) use a form of rely-guarantee (RG) reasoning (Jones 1983), a well-known method for verification of concurrent programs. A successful analysis proves that any execution of operations over replicated data, under a given synchronization protocol, maintains the given invariants.

The later steps of translating synchronization into efficient implementations in a geo-replicated model is also an open question; the challenge is that synchronous operations may hinder the latency advantage of asynchronous operations. There are different ways to implement a synchronous operation, each with own cost and complexity. From a performance perspective, there is no single best concurrency control protocol, since this will depend on dynamic characteristics of the workload, namely, on how often updates are blocked (contention) vs. how often synchronization are acquired (overhead). Combining static and dynamic analysis improves the reliability of the system. A future research direction would be to develop profiling or monitoring tools in order to measure the efficiency of the concurrency control protocol under different workloads and heuristics to improve it.

Another interesting direction is to study how different synchronization protocols perform under different failure models, in particular Byzantine failures. Failures including individual server failures, data center failures, and network failures are common in large-scale systems. A geo-replicated system must handle these failures. Some geo-replicated systems like Walter (Sovran et al. 2011) rely on crash-recovery model to handle failures. In the case of a primary replica's failure, a Paxos-based reconfiguration service can be used to replace the faulty process. After recovery, it must synchronize with the replacement replica to take over its responsibility. Finding solutions that can efficiently capture non-deterministic behavior of a geo-replicated system would be a good direction.

Cross-References

- ▶ Achieving Low Latency Transactions for Geo-Replicated Storage with Blotter
- ▶ Conflict-Free Replicated Data Types CRDTs
- ▶ Database Consistency Models
- ▶ TARDIS: A Branch-and-Merge Approach to Weak Consistency
- ▶ Weaker Consistency Models/Eventual Consistency

References

- Bailis P, Fekete A, Ghodsi A, Hellerstein JM, Stoica I (2013) HAT, not CAP: towards highly available transactions. In: Proceedings of the 14th USENIX conference on hot topics in operating systems, HotOS'13. USENIX Association, Berkeley, pp 24–24. <http://dl.acm.org/citation.cfm?id=2490483.2490507>
- Bailis P, Fekete A, Franklin MJ, Ghodsi A, Hellerstein JM, Stoica I (2015) Coordination avoidance in database systems. *PVLDB* 8:185–196
- Baker J, Bond C, Corbett JC, Furman J, Khorlin A, Larson J, Leon JM, Li Y, Lloyd A, Yushprakh V (2011) Megastore: providing scalable, highly available storage for interactive services. In: Proceedings of the conference on innovative data system research (CIDR), pp 223–234. http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf
- Balegas V, Preguiça N, Rodrigues R, Duarte S, Ferreira C, Najafzadeh M, Shapiro M (2015) Putting consistency back into eventual consistency. In: European conference on computer system (EuroSys), Bordeaux, pp 6:1–6:16. <https://doi.org/10.1145/2741948.2741972>, <http://dl.acm.org/citation.cfm?doid=2741948.2741972>
- Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R (2008) Pnutes: Yahoo!'s hosted data serving platform. *Proc VLDB Endow* 1(2):1277–1288. <https://doi.org/10.1145/1454159.1454167>
- Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman J, Ghemawat S, Gubarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D (2012) Spanner: Google's globally-distributed database. In: Symposium on operating systems design and implementation (OSDI), Usenix, Hollywood, pp 251–264
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. In: Symposium on Operating systems Principles (SOSP). Operating systems review, vol 41. Association for Computing Machinery, Stevenson, Washington, pp 205–220
- Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33(2):51–59. <https://doi.org/10.1145/564585.564601>
- Gotsman A, Yang H, Ferreira C, Najafzadeh M, Shapiro M (2016) 'Cause I'm strong enough: reasoning about consistency choices in distributed systems. In: Symposium on principles of programming languages (POPL), St. Petersburg
- Gray J, Lamport L (2006) Consensus on transaction commit. *Trans Database Syst* 31(1):133–160. <http://portal.acm.org/citation.cfm?id=1132863.1132867>

- Gray J, Reuter A (1993) Transaction processing: concepts and techniques. Morgan Kaufmann, San Francisco. ISBN:1-55860-190-2
- Guy R, Heidemann JS, Mak W, Popek GJ, Rothmeier D (1990) Implementation of the Ficus replicated file system. In: USENIX conference proceedings, pp 63–71
- Herlihy M, Wing J (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492. <https://doi.org/10.1145/78969.78972>
- Johnson PR, Thomas RH (1976) The maintenance of duplicate databases. Internet request for comments RFC 677. Information sciences institute. <http://www.rfc-editor.org/rfc.html>
- Jones CB (1983) Specification and design of (parallel) programs. In: IFIP congress, North-Holland
- Kaki G, Nagar K, Najafzadeh M, Jagannathan S (2018) Alone together: compositional reasoning and inference for weak isolation. In: Proceedings of the 45rd annual ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL'18
- Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. SIGOPS Oper Syst Rev 44(2):35–40. <https://doi.org/10.1145/1773912.1773922>
- Li C, Porto D, Clement A, Gehrke J, Preguiça N, Rodrigues R (2012) Making geo-replicated systems fast as possible, consistent when necessary. In: Symposium on operating systems design and implementation (OSDI), Hollywood, pp 265–278
- Li C, Leitão J, Clement A, Preguiça N, Rodrigues R, Vafeiadis V (2014) Automating the choice of consistency levels in replicated systems. In: Proceedings of the 2014 USENIX conference on USENIX annual technical conference, USENIX ATC'14. USENIX Association, Berkeley, pp 281–292. <http://dl.acm.org/citation.cfm?id=2643634.2643664>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In: Symposium on operating system principles (SOSP). Association for Computing Machinery, Cascais, pp 401–416. <https://doi.org/10.1145/2043556.2043593>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Networked systems design and implementation (NSDI), Lombard, pp 1–14
- Papadimitriou CH (1979) The serializability of concurrent database updates. J ACM 26(4):631–653. <http://portal.acm.org/citation.cfm?id=322154.322158>
- Popek G, Walker B, Chow J, Edwards D, Kline C, Rudisin G, Thiel G (1981) Locus: a network transparent, high reliability distributed system. In: Symposium on operating systems principles (SOSP). ACM, pp 169–177
- Ratner D, Reiher P, Popek G (1999) Roam: a scalable replication system for mobile computing. In: International workshop on database & expert systems applications (DEXA). IEEE Computer Society, Los Alamitos, pp 96–104. <http://doi.ieeecomputersociety.org/10.1109/DEXA.1999.795151>
- Satyanarayanan M, Kistler JJ, Kumar P, Okasaki ME, Siegel EH, Steere DC (1990) Coda: a highly available file system for a distributed workstation environment. IEEE Trans Comput 39:447–459
- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) Conflict-free replicated data types. In: Défago X, Petit F, Villain V (eds) International symposium on stabilization, safety, and security of distributed Systems (SSS). Lecture notes in computer science, vol 6976. Springer, Grenoble, pp 386–400
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Symposium on operating systems principles (SOSP). Association for Computing Machinery, Cascais, pp 385–400
- Terry DB, Theimer MM, Petersen K, Demers AJ, Spreitzer MJ, Hauser CH (1995) Managing update conflicts in Bayou, a weakly connected replicated storage system. In: Symposium on operating systems principles (SOSP). ACM SIGOPS. ACM Press, Copper Mountain, pp 172–182. <http://www.acm.org/pubs/articles/proceedings/ops/224056/p172-terry/p172-terry.pdf>
- Terry DB, Prabhakaran V, Kotla R, Balakrishnan M, Aguilera MK, Abu-Libdeh H (2013) Consistency-based service level agreements for cloud storage. In: SOSP
- Wang AI, Reiher P, Bagrodia R, Kuenning G (2002) Understanding the behavior of the conflict-rate metric in optimistic peer replication. In: Proceedings of the 13th international workshop on database and expert systems applications, pp 757–761. <https://doi.org/10.1109/DEXA.2002.1045989>
- Wittie MP, Pejovic V, Deek L, Almeroth KC, Zhao BY (2010) Exploiting locality of interest in online social networks. In: Proceedings of the 6th international conference, Co-NEXT'10. ACM, New York, pp 25:1–25:12. <https://doi.org/10.1145/1921168.1921201>

Geo-Scale Transaction Processing

Faisal Nawab

University of California, Santa Cruz, CA, USA

Synonyms

[Geo-distributed transaction processing](#); [Geo-replicated transaction processing](#); [Global-scale transaction processing](#)

Definitions

Geo-Scale Transaction Processing considers the processing of transactions on nodes that are separated by wide-area links.

Overview

Replication and distribution of data across nodes have been used for various objectives, such as fault tolerance, load balancing, read availability, and others. This practice dates back to the early days of computing (Kemme et al. 2010; Bernstein and Goodman 1981) and continues to develop to accommodate the development and advancement of new computing technologies. The availability of publicly accessible cloud resources that are dispersed around the world has allowed the replication and distribution of data across large distances, potentially covering many continents. This is denoted a geo-scale deployment and allows achieving higher levels of the objectives of replication and distribution. For example, geo-scale deployments can tolerate data center-scale failures and disasters that cover large geographic regions. Also, geo-scale deployments can provide faster access to users across the globe. In addition to the classical objectives of replication and distribution, geo-scale deployment allows a better utilization of cloud resources, as it enables the use of resources from multiple cloud providers and cloud locations. Typically, different cloud providers and locations offer varying services with diverse characteristics. A geo-scale deployment allows leveraging heterogeneous cloud resources to optimize various performances and monetary objectives.

With the increasing distance between nodes, the data management tasks to provide access to data become more challenging. One essential data management task for access is transaction processing. A transaction is an abstraction of data that consists of a collection of operations, typically reads and writes, and that is guaranteed to execute with certain defined properties. These properties include ones about consistency, recoverability, and others, and each property has many flavors, defining the strictness of the provided guarantees. The cost of providing various guarantees increases significantly in geo-scale deployment due to the wide-area link between nodes. This is due to the communication needed between nodes to coordinate access while preserving defined guarantees. This has stimulated interest in developing new transaction processing technologies for geo-scale deployments that ameliorate the cost of wide-area communication. This includes new designs that achieve classical guarantees more efficiently in geo-scale environments and new explorations on transactional access guarantees that are more feasible in geo-scale environments.

Key Research Findings

Geo-Scale Transaction Processing includes work that covers various consistency guarantees. Next is an overview of key research findings for different levels of transactional consistency guarantees. Related research findings on Geo-Scale Transaction Processing follows.

Strongly Consistent Transactions

Serializability (Bernstein et al. 1987) is a strong consistency guarantee that ensures that the outcome of concurrent transaction execution is equivalent to some serial execution of these transactions. It has been extensively studied in the context of database systems and is considered one of the strongest guarantees on consistency that enables abstracting out the complexities of concurrency and replication, thus ridding users from thinking about these complexities. However, serializability is a strong consistency guarantee that requires extensive coordination between nodes executing transactions and nodes hosting data copies. This makes it especially challenging in a geo-scale deployment, where coordination is across wide-area links. To overcome the expensive cost of wide-area coordination, a redesign of well-known transaction processing protocols and approaches has been conducted in addition to the

design of novel transaction processing protocols specifically for geo-scale deployments.

Paxos (Lamport 1998) is a fault-tolerant consensus protocol that has been used widely to implement transaction processing systems that provide strong guarantees. In these implementations, Paxos provides a state machine replication (SMR) component that guarantees the totally ordered execution of requests (e.g., transactions). Due to its wide use in data management systems, there have been many attempts to build Paxos-based protocols for Geo-Scale Transaction Processing, such as megastore (Baker et al. 2011), Paxos-CP (Patterson et al. 2012), Egalitarian Paxos (Moraru et al. 2013), and MDCC (Kraska et al. 2013; Pang et al. 2014).

Megastore (Baker et al. 2011) uses Paxos to execute transactions on multiple nodes in different data centers. Paxos is used to assign transactions to log positions and to make all nodes agree on the assigned order. Then, each node executes the transactions according to the assigned order. This guarantees that all nodes maintain a consistent copy of data, since they all execute transactions in the same order. Assigning the position for a request requires a round of communication, at least, between a Paxos leader and other nodes. Paxos-CP (Patterson et al. 2012) extends megastore by increasing the utilization of Paxos communication. Specifically, in the presence of concurrent transactions, Paxos-CP allows *combining* transactions in the same log position rather than starting a new round of messaging.

MDCC (Kraska et al. 2013; Pang et al. 2014) leverages a follow-up Paxos variant, called Fast Paxos (Lamport 2006) that exhibits better performance characteristics for geo-scale deployments compared to the original Paxos protocol. Fast Paxos, unlike the original Paxos protocol, is a leaderless protocol, meaning that leader election, a major component of Paxos' design, is not needed. Therefore, for users at different locations, it is possible for all of them to process transactions without having to go through a leader. This comes at the cost of larger quorums (i.e., groups of nodes) that are needed to process requests. In Paxos, a simple majority suffices to

process a request (from a leader), whereas Fast Paxos requires a fast quorum that is typically larger than a majority of nodes. Egalitarian Paxos (EPaxos) (Moraru et al. 2013) proposes a new variant of Paxos that combine many attractive features for geo-scale deployments. First, EPaxos is a leaderless protocol; thus, it enjoys the benefits of leaderless protocols in comparison to the original Paxos protocol as we discuss previously for Fast Paxos. Additionally, EPaxos reduces the quorum size to be smaller than what is typically needed by MDCC and achieves the optimal quorum size (i.e., three nodes) of the widely used case of five nodes. EPaxos also utilizes many useful features to increase concurrency, such as allowing nonconflicting operations to execute concurrently. This is done via a multidimensional log, similar to generalized Paxos (Lamport 2005) rather than a simple log of requests that enforce ordering events even if they do not conflict.

Paxos was also shown to be suitable to act as a synchronous communication layer integrated with a transaction commit protocol (Corbett et al. 2012; Glendenning et al. 2011; Mahmoud et al. 2013). For example, Spanner (Corbett et al. 2012) and Replicated Commit (Mahmoud et al. 2013) commit using variants of two-phase commit (2PC) and strict two-phase locking (S2PL) and leverage Paxos to replicate across data centers. Spanner partitions data and assigns a leader for each partition. Each partition is synchronously replicated, for fault tolerance, to a set of replicas using Paxos. Transactions that span more than one partition are executed by coordinating between the corresponding partition leaders. This architecture separates consistency from durability, allowing optimizing each in isolation. Also, transactions that access a single partition are not subject to the multi-partition coordination. Replicated Commit executes transactions by getting votes from a majority of data centers. A data center's positive vote is a promise to not allow the execution of conflicting transactions. Once a majority positive votes are collected, the transaction proceeds to commit. Replicated Commit conflates consistency and durability, which reduces the amount of redundant messages. Additionally,

since there are no leaders, Replicated Commit exhibits better load balancing characteristics. However, a majority vote is always needed for all transactions, even ones that access a single partition.

To ameliorate the cost of wide-area communication, proactive coordination techniques were developed that exchange data continuously to enable early detection of conflicts (Nawab et al. 2013, 2015b). Helios (Nawab et al. 2015b), a work based on proactive coordination, finds the optimal transaction commit latency via a theoretical model of coordination and proposes a protocol design that achieves the optimal latency. Helios shows that for any pair of nodes, the sum of their transaction latency cannot be lower than the round-trip time latency between them (Nawab et al. 2015b).

Timestamp-based and time synchronization techniques have been explored for Geo-Scale Transaction Processing systems (Corbett et al. 2012; Du et al. 2013a,b). Although the distance between data centers may be large, Spanner (Corbett et al. 2012) shows that accurate time synchronization is achievable using specialized infrastructure such as atomic clocks and GPS. Without specialized hardware, accurate time synchronization is challenging. However, other geo-scale systems explored the use of loosely synchronized clock methods (Du et al. 2013a,b; Nawab et al. 2015b).

Weak and Relaxed Transactional Semantics

A major source of performance overhead in Geo-Scale Transaction Processing protocols is due to the coordination between nodes to guarantee consistent outcomes. To overcome the overhead of coordination, many lines of work investigates the trade-off between consistency and performance. In such investigations, consistency is relaxed or weakened to enable higher levels of performance. Central to this study is to understand the implications of relaxing and weakening consistency and what this means to the users and application developers. For example, application developers may have to cope with anomalies that arise due to the weaker consistency levels.

An extreme point in the performance-consistency trade-off spectrum is the weakening of consistency and isolation guarantees in favor of performance (Cooper et al. 2008; DeCandia et al. 2007; Bailis and Ghodsi 2013). Adopting this approach entails weakening the guarantees of how conflicts between requests are handled. For example, eventual consistency delays the arbitration of conflicts and only guarantees that conflicts are resolved eventually – thus creating the possibility of temporary inconsistency. Also, this approach may entail weakening the access abstractions, such as providing single-key access abstractions rather than multi-object transactions. This limits the interface to application developers; however, it is easier to manage and enables higher levels of performance.

In addition to the two extremes in the consistency-performance trade-off spectrum, there are consistency levels in between that exhibit various performance and consistency characteristics. Exploring consistency levels in the context of Geo-Scale Transaction Processing aimed to overcome the high cost of wide-area coordination while providing some consistency guarantees for application developers. To this end, there have been many efforts to study weaker consistency guarantees and then extending them to adapt to geo-scale environments. Causal consistency, that is inspired by the causal ordering principle (Lamport 1978), is one such consistency level. Causal consistency guarantees that events (e.g., transactions) are ordered at all nodes according to their causal relations. Causal relations in this context denote ordering guarantees between events at the same node (total order), ordering of events at a node before any subsequent received events (happens-before relation), and ordering due to transitive total order or happens-before causal relations. There have been many Geo-Scale Transaction Processing systems based on causal consistency (Lloyd et al. 2011, 2013; Bailis et al. 2013; Nawab et al. 2015a; Du et al. 2013a). An example is COPS (Lloyd et al. 2011, 2013) which extends causal consistency with a convergence guarantee and call it Causal+ consistency. COPS provides

a read-only transactional interface that guarantees Causal+ execution via a two-round protocol.

Another relaxed consistency guarantee that was explored for Geo-Scale Transaction Processing is snapshot isolation (SI) (Berenson et al. 1995) which guarantees that a transaction reads a consistent snapshot of data and that write-write conflicts are detected. Geo-Scale Transaction Processing systems have been built by extending the concept of SI (Sovran et al. 2011; Daudjee and Salem 2006; Lin et al. 2005, 2007; Du et al. 2013b). An example is Walter (Sovran et al. 2011) which builds upon SI and proposes Parallel SI (PSI). PSI relaxes the snapshot and write-write conflict definitions to allow replicating data asynchronously between data centers, which is not possible in SI.

Coordination-Free Processing

Another approach to tackle the consistency-performance trade-off is to explore classes of transactions that can be executed without coordination while maintaining consistency. This approach is called coordination-free processing and has been explored in the context of Geo-Scale Transaction Processing (Bailis et al. 2014; Roy et al. 2015; Zhang et al. 2013). An example of this work is transaction chains (Zhang et al. 2013) that analyze an application's transactions and build an execution plan for distributed transactions with the objective of fast response time, equivalent to executing at a single data center with no wide-area communication.

Data Placement

The topology of the geo-scale system plays a significant role in transaction performance. Therefore, placement of data and workers on carefully chosen data centers has the potential of optimizing performance. Placement has been tackled in the context of geo-scale systems (Zakhary et al. 2016, 2018; Wu et al. 2013; Vulimiri et al. 2015; Agarwal et al. 2010; Endo et al. 2011; Pu et al. 2015; Shankaranarayanan et al. 2014; Sharov et al. 2015). Some of these placement frameworks specifically target optimizing placement for Geo-Scale Transaction Processing (Zakhary et al. 2016, 2018; Sharov et al. 2015). For ex-

ample, (Sharov et al. 2015) propose an optimization formulation for placement with an objective of minimizing latency for leader-based transaction processing systems such as Spanner (Corbett et al. 2012). Zakhary et al. (2016, 2018) propose placement formulations for quorum-based transaction processing systems such as Paxos-based systems.

Managing the Consistency-Performance Trade-Off

Rather than a fixed Geo-Scale Transaction Processing protocol, there have been studies on dynamic protocols that adapt their consistency level and protocol characteristics based on the current performance (Wu et al. 2015; Terry et al. 2013; Ardekani and Terry 2014). For example, CosTLO (Wu et al. 2015) introduces redundancy in messaging to lower the variance of latency. Another example is Pileus and Tuba (Terry et al. 2013; Ardekani and Terry 2014) that enables applications to dynamically control the consistency level by declaring consistency and latency priorities. According to these priorities, the system adapts its protocol.

Conclusion

Geo-Scale Transaction Processing provides easy-to-use abstractions of access to data that is globally replicated or distributed. One of the major challenges faced by Geo-Scale Transaction Processing is the large wide-area latency between nodes. To overcome this challenge, redesigns of transaction processing systems are proposed, and studies of the consistency-performance trade-off are conducted. These approaches are shown to optimize the performance of Geo-Scale Transaction Processing by considering the wide-area latency as the main bottleneck.

Examples of Application

Geo-Scale Transaction Processing is used by web and cloud applications.

Future Directions for Research

An avenue of future research is to expand beyond data centers and build Geo-Scale Transaction Processing protocols that utilize edge resources.

Cross-References

- ▶ [Coordination Avoidance](#)
- ▶ [Geo-Replication Models](#)
- ▶ [Weaker Consistency Models/Eventual Consistency](#)

References

- Agarwal S, Dunagan J, Jain N, Saroiu S, Wolman A, Bhogan H (2010) Volley: automated data placement for geo-distributed cloud services. In: Proceedings of the 7th USENIX conference on networked systems design and implementation, USENIX association, NSDI'10, Berkeley, pp 2–2. <http://dl.acm.org/citation.cfm?id=1855711.1855713>
- Ardekani MS, Terry DB (2014) A self-configurable geo-replicated cloud storage system. In: Proceedings of the 11th USENIX conference on operating systems design and implementation, USENIX association, OSDI'14, Berkeley, pp 367–381. <http://dl.acm.org/citation.cfm?id=2685048.2685077>
- Bailis P, Ghodsi A (2013) Eventual consistency today: limitations, extensions, and beyond. Queue 11(3):20:20–20:32. <http://doi.acm.org/10.1145/2460276.2462076>
- Bailis P, Ghodsi A, Hellerstein JM, Stoica I (2013) Bolt-on causal consistency. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD'13. ACM, New York, pp 761–772. <http://doi.acm.org/10.1145/2463676.2465279>
- Bailis P, Fekete A, Franklin MJ, Ghodsi A, Hellerstein JM, Stoica I (2014) Coordination avoidance in database systems. Proc VLDB Endow 8(3):185–196. <https://doi.org/10.14778/2735508.2735509>
- Baker J, Bond C, Corbett JC, Furman JJ, Khorlin A, Larson J, Leon J, Li Y, Lloyd A, Yushprakh V (2011) Megastore: providing scalable, highly available storage for interactive services. In: CIDR 2011, Fifth biennial conference on innovative data systems research, Asilomar, pp 223–234, 9–12 Jan 2011. Online Proceedings. http://cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf
- Berenson H, Bernstein P, Gray J, Melton J, O’Neil E, O’Neil P (1995) A critique of ansi SQL isolation levels, pp 1–10. <http://doi.acm.org/10.1145/223784.223785>
- Bernstein PA, Goodman N (1981) Concurrency control in distributed database systems. ACM Comput Surv (CSUR) 13(2):185–221
- Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Reading
- Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R (2008) Pnuts: Yahoo!’s hosted data serving platform. Proc VLDB Endow 1(2):1277–1288. <https://doi.org/10.14778/1454159.1454167>
- Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Guibarev A, Heiser C, Hochschild P, Hsieh W, Kanthak S, Kogan E, Li H, Lloyd A, Melnik S, Mwaura D, Nagle D, Quinlan S, Rao R, Rolig L, Saito Y, Szymaniak M, Taylor C, Wang R, Woodford D (2012) Spanner: Google’s globally-distributed database, pp 251–264. <http://dl.acm.org/citation.cfm?id=2387880.2387905>
- Daudjee K, Salem K (2006) Lazy database replication with snapshot isolation. In: Proceedings of the 32nd international conference on very large data bases, VLDB Endowment, VLDB’06, pp 715–726. <http://dl.acm.org/citation.cfm?id=1182635.1164189>
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon’s highly available key-value store. In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP’07. ACM, New York, pp 205–220. <http://doi.acm.org/10.1145/1294261.1294281>
- Du J, Elnikety S, Roy A, Zwaenepoel W (2013a) Orbe: scalable causal consistency using dependency matrices and physical clocks. In: Proceedings of the 4th annual symposium on cloud computing, SOCC’13. ACM, New York, pp 11:1–11:14. <http://doi.acm.org/10.1145/2523616.2523628>
- Du J, Elnikety S, Zwaenepoel W (2013b) Clock-si: Snapshot isolation for partitioned data stores using loosely synchronized clocks. In: Proceedings of the 2013 IEEE 32nd international symposium on reliable distributed systems, SRDS’13. IEEE Computer Society, Washington, pp 173–184, <https://doi.org/10.1109/SRDS.2013.26>
- Endo PT, de Almeida Palhares AV, Pereira NN, Goncalves GE, Sadok D, Kelner J, Melander B, Mangs JE (2011) Resource allocation for distributed cloud: concepts and research challenges. IEEE Netw 25(4):42–46. <https://doi.org/10.1109/MNET.2011.5958007>
- Glendenning L, Beschastník I, Krishnamurthy A, Anderson T (2011) Scalable consistency in scatter. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP’11. ACM, New York, pp 15–28. <http://doi.acm.org/10.1145/2043556.2043559>
- Kemme B, Jimenez-Peris R, Patino-Martinez M (2010) Database replication. Synth Lect Data Manage 2(1):1–153. <http://www.morganclaypool.com/doi/abs/10.2200/S00296ED1V01Y201008DTM007>

- Kraska T, Pang G, Franklin MJ, Madden S, Fekete A (2013) MDCC: multi-data center consistency. In: Proceedings of the 8th ACM European conference on computer systems, EuroSys'13. ACM, New York, pp 113–126. <http://doi.acm.org/10.1145/2465351.2465363>
- Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. <http://doi.acm.org/10.1145/359545.359563>
- Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169. <http://doi.acm.org/10.1145/279227.279229>
- Lamport L (2005) Generalized consensus and paxos. Technical report, MSR-TR-2005-33, Microsoft Research
- Lamport L (2006) Fast paxos. Distrib Comput 19(2):79–103
- Lin Y, Kemme B, Patiño Martínez M, Jiménez-Peris R (2005) Middleware based data replication providing snapshot isolation. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD'05. ACM, New York, pp 419–430. <http://doi.acm.org/10.1145/1066157.1066205>
- Lin Y, Kemme B, Patino-Martinez M, Jimenez-Peris R (2007) Enhancing edge computing with database replication. In: Proceedings of the 26th IEEE international symposium on reliable distributed systems, SRDS'07. IEEE Computer Society, Washington, pp 45–54. <http://dl.acm.org/citation.cfm?id=1308172.1308219>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 401–416. <http://doi.acm.org/10.1145/2043556.2043593>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Proceedings of the 10th USENIX conference on networked systems design and implementation, NSDI'13. USENIX Association, Berkeley, pp 313–328. <http://dl.acm.org/citation.cfm?id=2482626.2482657>
- Mahmoud H, Nawab F, Pucher A, Agrawal D, El Abbadi A (2013) Low-latency multi-datacenter databases using replicated commit. Proc VLDB Endow 6(9):661–672. <https://doi.org/10.14778/2536360.2536366>
- Moraru I, Andersen DG, Kaminsky M (2013) There is more consensus in Egalitarian parliaments. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP'13. ACM, New York, pp 358–372. <http://doi.acm.org/10.1145/2517349.2517350>
- Nawab F, Agrawal D, El Abbadi A (2013) Message futures: fast commitment of transactions in multi-datacenter environments. In: CIDR 2013, sixth biennial conference on innovative data systems research, Asilomar, 6–9 Jan 2013. Online Proceedings. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper103.pdf
- Nawab F, Arora V, Agrawal D, El Abbadi A (2015a) Chariots: a scalable shared log for data management in multi-datacenter cloud environments. In: Proceedings of the 18th international conference on extending database technology, EDBT 2015, Brussels, 23–27 Mar 2015, pp 13–24. <https://doi.org/10.5441/002/edbt.2015.03>
- Nawab F, Arora V, Agrawal D, El Abbadi A (2015b) Minimizing commit latency of transactions in geo-replicated data stores. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD'15. ACM, New York, pp 1279–1294. <http://doi.acm.org/10.1145/2723372.2723729>
- Pang G, Kraska T, Franklin MJ, Fekete A (2014) Planet: making progress with commit processing in unpredictable environments. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD'14. ACM, New York, pp 3–14. <http://doi.acm.org/10.1145/2588555.2588558>
- Patterson S, Elmore AJ, Nawab F, Agrawal D, El Abbadi A (2012) Serializability, not serial: concurrency control and availability in multi-datacenter datastores. Proc VLDB Endow 5(11):1459–1470. <https://doi.org/10.14778/2350229.2350261>
- Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. In: Proceedings of the 2015 ACM conference on special interest group on data communication, SIGCOMM'15. ACM, New York, pp 421–434. <http://doi.acm.org/10.1145/2785956.2787505>
- Roy S, Kot L, Bender G, Ding B, Hojjat H, Koch C, Foster N, Gehrke J (2015) The homeostasis protocol: avoiding transaction coordination through program analysis. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD'15. ACM, New York, pp 1311–1326. <http://doi.acm.org/10.1145/2723372.2723720>
- Shankaranarayanan PN, Sivakumar A, Rao S, Tawar-malani M (2014) Performance sensitive replication in geo-distributed cloud datastores. In: Proceedings of the 2014 44th annual IEEE/IFIP international conference on dependable systems and networks, DSN'14. IEEE Computer Society, Washington, pp 240–251. <https://doi.org/10.1109/DSN.2014.34>
- Sharov A, Shraer A, Merchant A, Stokely M (2015) Take me to your leader!: online optimization of distributed storage configurations. Proc VLDB Endow 8(12):1490–1501. <https://doi.org/10.14778/2824032.2824047>
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 385–400. <http://doi.acm.org/10.1145/2043556.2043592>
- Terry DB, Prabhakaran V, Kotla R, Balakrishnan M, Aguilera MK, Abu-Libdeh H (2013) Consistency-based service level agreements for cloud storage. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP'13. ACM, New York, pp 309–324. <http://doi.acm.org/10.1145/2517349.2522731>

- Vulimiri A, Curino C, Godfrey PB, Jungblut T, Padhye J, Varghese G (2015) Global analytics in the face of bandwidth and regulatory constraints. In: Proceedings of the 12th USENIX conference on networked systems design and implementation, NSDI'15. USENIX Association, Berkeley, pp 323–336. <http://dl.acm.org/citation.cfm?id=2789770.2789793>
- Wu Z, Butkiewicz M, Perkins D, Katz-Bassett E, Madhyastha HV (2013) Spanstore: cost-effective geo-replicated storage spanning multiple cloud services. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP'13. ACM, New York, pp 292–308. <http://doi.acm.org/10.1145/2517349.2522730>
- Wu Z, Yu C, Madhyastha HV (2015) Costlo: cost-effective redundancy for lower latency variance on cloud storage services. In: Proceedings of the 12th USENIX conference on networked systems design and implementation, NSDI'15. USENIX Association, Berkeley, pp 543–557. <http://dl.acm.org/citation.cfm?id=2789770.2789808>
- Zakhary V, Nawab F, Agrawal D, El Abbadi A (2016) Db-risk: the game of global database placement. In: Proceedings of the 2016 international conference on management of data, SIGMOD'16. ACM, New York, pp 2185–2188. <http://doi.acm.org/10.1145/2882903.2899405>
- Zakhary V, Nawab F, Agrawal D, El Abbadi A (2018) Global-scale placement of transactional data stores. In: Proceedings of the 2018 international conference on extending database technology, EDBT'18
- Zhang Y, Power R, Zhou S, Sovran Y, Aguilera MK, Li J (2013) Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, SOSP'13. ACM, New York, pp 276–291. <http://doi.acm.org/10.1145/2517349.2522729>

Geosocial Data

- ▶ Spatio-social Data

Geospatial Data Integration

- ▶ Spatial Data Integration

Geo-textual Data

- ▶ Spatio-textual Data

Global-Scale Transaction Processing

- ▶ Geo-scale Transaction Processing

GPU-Based Hardware Platforms

Johns Paul¹, Bingsheng He², and Chiew Tong Lau¹

¹Nanyang Technological University, Singapore, Singapore

²National University of Singapore, Singapore, Singapore

Definitions

GPU, SIMD, single-package platforms, multi-package platforms GPU-based hardware platforms are platforms that usually use GPUs in conjunction with CPUs to accelerate specific tasks. GPUs have evolved as a powerful accelerator for processing huge amounts of data in parallel.

Overview

This chapter details the architectural design and internal working of GPU-based hardware platforms. For this, we broadly classify GPU-based hardware platforms into two categories: single-package and multi-package. We then detail the architectural differences as well as the advantages and limitations of each category.

GPU

Graphics processing units (GPUs) were initially designed to accelerate the rendering of images for video editing and gaming. However, more recently, GPUs have evolved as a powerful accelerator for processing huge amounts of data

in parallel. Similar to CPUs, GPUs also contains processing cores, registers, multiple levels of cache, and a global memory unit. However, GPUs differ from CPUs in terms of the design of each one of these components. Instead of a small number of complex cores available on the CPUs, GPUs use large number of relatively simpler cores that work in a single instruction multiple data (SIMD) fashion to process large amounts of data. Modern GPUs contain more than 100x the number of cores in modern CPUs. This allows GPUs to execute the same operation on a large number of data items in parallel. GPUs are able to fit such large number of cores in a single silicon die due to the relative simplicity of each individual core. These cores lack complex components such as branch prediction unit and even have much smaller cache than CPUs.

The GPU cores also have access to a much larger number of registers and a higher bandwidth global memory unit, which is usually smaller (in size) than the main memory available on CPUs. The high bandwidth memory (HBM) units available in modern GPUs can provide close to 10x the bandwidth of main memory accessible to CPUs. This availability of larger number of registers and the much higher global memory bandwidth of GPUs ensures fast data access to the large number of parallel cores, thus making it possible to achieve much better throughput than CPUs. Figure 1 shows an overview of the internal architecture of GPUs.

The high level of parallelism provided by GPUs makes them an attractive platform for many high-performance computing tasks (Owens et al. 2007, 2008; Nickolls and Dally 2010). Over the past few years, GPUs have become popular for applications like 3D rendering (Kruger and Westermann 2003; Cates et al. 2018; Liu et al. 2004), deep-learning (Chetlur et al. 2014; Bergstra et al. 2011), online analytical processing (Paul et al. 2016; Heimel et al. 2013), and weather modeling (Shimokawabe et al. 2010; Mielikainen et al. 2012). Such applications often involve executing the same instruction on large number of data entries in parallel and have minimal amount of incoherent branching. Hence, they benefit from the huge parallelism

provided by GPUs, without encountering any of the penalty associated with divergent execution.

In spite of being able to achieve high performance for parallel tasks, GPUs are not suitable for running generic tasks like an operating system. Hence, any system that takes advantage of GPUs for accelerating tasks needs a CPU component to run generic tasks like schedulers and resource managers. Hence, GPU-based hardware platforms are platforms that usually use GPUs in conjunction with CPUs to accelerate specific tasks. These hardware platforms can broadly be classified into single-package and multi-package platforms, based on the level of integration between the CPU and GPU. Single-package platforms contain a GPU and a CPU on a single semiconductor package (Wikipedia 2011). Multi-package platforms are those that contain a CPU and a GPU in two separate packages, and the communication between the two happens over interconnections like PCIe or NVLink.

G

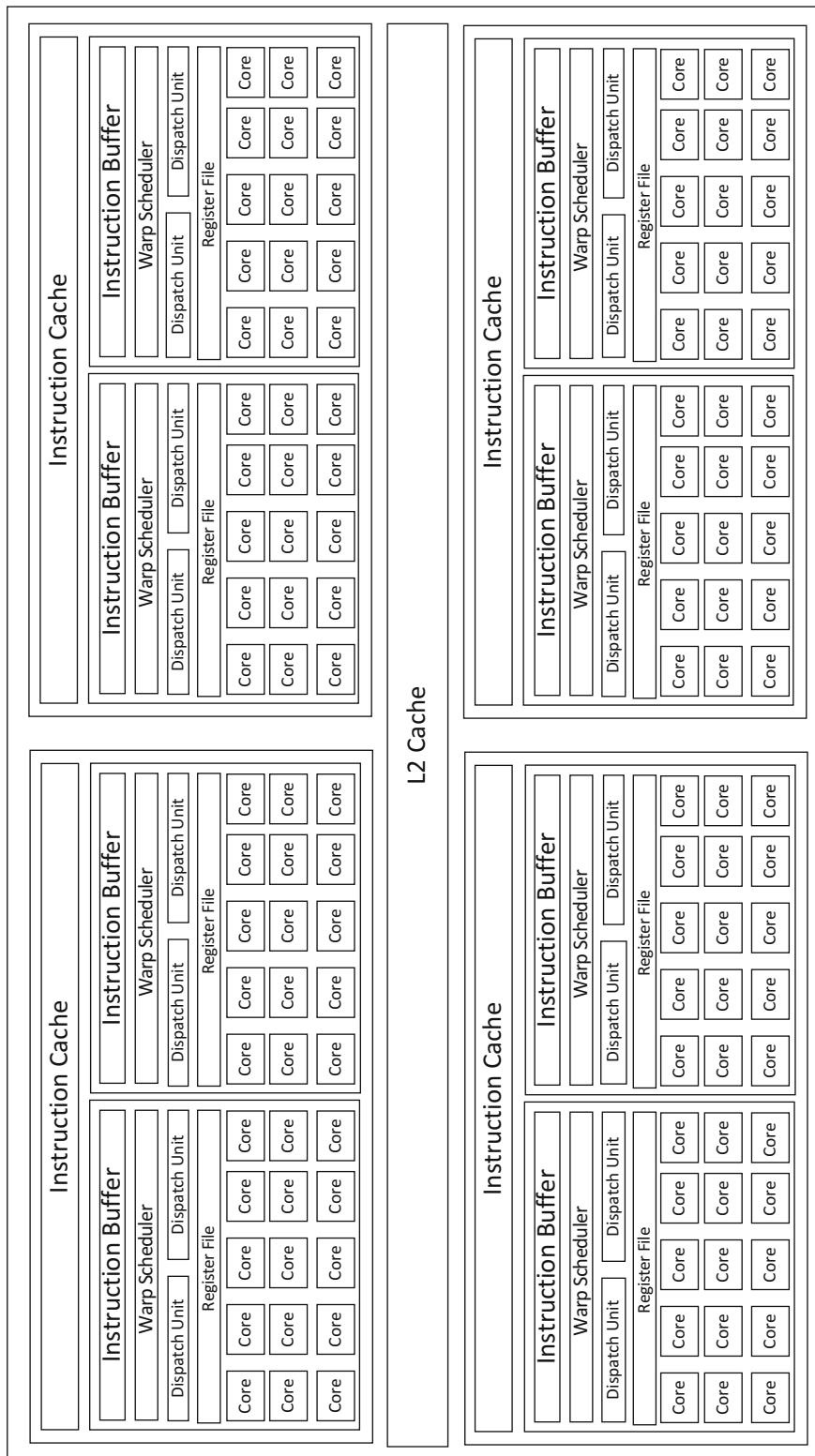
Single Package

In a single-package platform, the CPU and the GPU components are colocated within the same semiconductor package. Such systems can contain a single silicon die or multiple silicon dies interconnected using a silicon bridge. These single-package platforms can further be divided into (1) mobile platforms used in small handheld devices and (2) desktop platforms used in larger systems for more high-performance tasks.

Mobile Platforms

Most SoCs used in mobile devices these days contain GPUs in some form. Some of the most common examples include NVIDIA Tegra (NVIDIA 2015), Qualcomm Snapdragon SoCs (Qualcomm 2017), and Apple's A series SoCs (Wikipedia 2017). The mobile GPU platforms are usually designed to minimize power consumption and are usually cooled passively due to space restrictions.

The latest Tegra X1 SoC from NVIDIA is their successor to the Tegra K1 SoC and contains an ARM CPU combined with an NVIDIA Maxwell



GPU-Based Hardware Platforms, Fig. 1 GPU internal architecture

GPU. The ARM CPU is a 64-bit A57 with four cores and 2MB of L2 cache, while the GPU is 256 core Maxwell generation GPU. The latest Snapdragon 835 SoC from Qualcomm contains the Qualcomm Kryo 280 CPU which is a 64-bit CPU, based on the ARM Cortex Technology. And the A11 SoC from Apple, which is its successor to the A10 SoC, contains 64-bit ARMv8 CPU coupled with a triple-core GPU designed by Apple.

Desktop Platforms

Desktop GPU platforms are generally much more capable than mobile platforms, as these devices can afford to consume much more power and are usually cooled by dedicated hardware. Some of the examples of the commercially available hardware platforms of this kind include AMD Ryzen Accelerated Processing Units (APUs) (AMD 2017a) and Intel's latest efforts to integrate AMD Radeon Graphics with its eighth-generation CPUs (AnandTech 2017) in the same physical package using EMIB technology.

The latest AMD Ryzen 7 2700U APU from AMD, which is their successor to the Fusion series APUs, consists of a quad-core CPU with up to 3.8 GHz boost clock and a Vega 10 GPU with ten CUs operating at up to 1.3 GHz. Finally, the latest announcement comes from Intel regarding their plans to combine an eighth-generation Intel CPU with AMD Radeon graphics. Unlike the other two approaches, Intel plans to interlink two separate silicon die (one containing an Intel CPU and the other containing an AMD GPU) into a single package using the embedded multi-die interconnect bridge (EMIB) technology, where the GPU and the CPU communicate over a high-speed silicon bridge.

Advantages and Limitations

The main advantage of such a single-package design is the high bandwidth and low latency of communication between the CPU and the GPU. Another advantage is that in many cases, due to the close integration between the hardware devices, the GPU is able to directly access data from the CPU like the contents of the L3 cache. Such systems also allow the use of a single unified memory unit (accessible to both CPU and GPU)

instead of having separate memory units for the CPU and GPU. This helps avoid the overhead of explicit movement of data from the main memory to the GPU global memory, as in the case of multi-package systems.

The main limitation of single-package systems comes in the form of higher heat generation and the limitations in terms of package size. Integrating CPUs and GPUs into the same silicon die or semiconductor package will lead to higher energy output, due to more high-performance components cramped into a small space. This can limit the capabilities of the system as a whole. Also mass production of very large semiconductor packages is a difficult endeavor, and hence the size of CPU or GPU that can fit within the unified system will be much more limited than multi-package systems leading to less powerful individual components.

G

Multi-package

Multi-package systems consist of GPUs and CPUs in two separate semiconductor packages on the same physical board or even on two separate physical boards. Examples of such systems include standalone GPU hardware that can be connected to CPUs over interconnects like NVLink and PCIe. Both AMD and NVIDIA release such standalone or discreet GPUs every year. In these systems, the GPUs contain separate physical global memory which stores all the data that can be processed by the GPU. Due to their much larger size and higher-power requirements, multi-package systems are not suitable for mobile devices and are usually restricted to desktop devices.

The latest P100 discreet GPU from NVIDIA (2016) contains more than 3500 individual cores that can operate at frequency of up to 1.4 GHz. The P100 also have two different variants: NVLink based and PCIe based. The PCIe-based model only supports the PCIe bus for communication, while the NVLink version also supports the use of NVLink interconnect which can provide up to 5x speed of the PCIe interconnect. The P100 also supports up to

16 GB of high bandwidth memory. The latest RX Vega 64 GPU AMD (2017b) from AMD on the other hand contains over 4000 cores operating at up to 1.4 GHz. They also contain up to 8 GB of high bandwidth memory.

Advantages and Limitations

The main advantage of the multi-package GPU platforms is that these systems can support much more powerful individual components when compared to single-package designs. This is because separate packages can be cooled more efficiently, and individual components (GPU and CPU) can be much larger in size when compared to single-package systems where both the components need to cramp into a single semiconductor package. Further, the use of interconnects like PCIe means that more GPUs can be added to the platform at a later stage (based on the CPU and available PCIe lanes), making such system more flexible and upgradeable when compared to single-package systems.

However, the multi-package systems suffer from major bottlenecks when it comes to actual performance. Such systems are usually bottlenecked by the speed of the interconnect between the CPU and the GPU. This is because any data that needs to be processed by the GPU needs to be transferred from the system main memory to the GPU global memory via the PCIe bus. Further, these discreet GPUs lack access to CPU cache and in many cases even lack direct access to the CPU main memory leading high communication overhead between the CPU and the GPU.

Cross-References

► Search and Query Accelerators

References

AMD (2017a) AMD Ryzen mobile. <http://www.amd.com/en/products/ryzen-processors-laptop>

- AMD (2017b) AMD Vega 64. <https://gaming.radeon.com/en/product/vega/radeon-rx-vega-64/>
- AnandTech (2017) Intel to create 8th generation CPUs with AMD Radeon graphics. <https://www.anandtech.com/show/12003/intel-to-create-new-8th-generation-cpus-with-amd-radeon-graphics-with-hbm2-using-emib>
- Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow IJ, Bergeron A, Bengio Y (2011) Theano: deep learning on GPUs with python. In: Big learn workshop, NIPS'11
- Cates JE, Lefohn AE, Whitaker RT (2018) GIST: an interactive, GPU-based level set segmentation tool for 3D medical images. *Med Image Anal* 8(3):217–231. <https://doi.org/10.1016/j.media.2004.06.022>
- Chetlur S, Woolley C, Vandermersch P, Cohen J, Tran J, Catanzaro B, Shelhamer E (2014) cuDNN: efficient primitives for deep learning. CoRR. <http://arxiv.org/abs/1410.0759>. <https://dblp.org/rec/bib/journals/corr/ChetlurWVCTCS14>
- Heimel M, Saecker M, Pirk H, Manegold S, Markl V (2013) Hardware-oblivious parallelism for in-memory column-stores. *Proc VLDB Endow* 6(9):709–720. <https://doi.org/10.14778/2536360.2536370>
- Kruger J, Westermann R (2003) Acceleration techniques for GPU-based volume rendering. In: Proceedings of the 14th IEEE visualization 2003 (VIS'03), IEEE computer society, Washington, DC, p 38. <https://doi.org/10.1109/VIS.2003.10001>
- Liu Y, Liu X, Wu E (2004) Real-time 3d fluid simulation on GPU with complex obstacles. In: Proceedings of the 12th Pacific conference on computer graphics and applications, PG 2004, pp 247–256. <https://doi.org/10.1109/PCCGA.2004.1348355>
- Mielikainen J, Huang B, Huang HLA, Goldberg MD (2012) Improved GPU/CUDA based parallel weather and research forecast (WRF) single moment 5-class (WSM5) cloud microphysics. *IEEE J Sel Top Appl Earth Obs Remote Sens* 5(4):1256–1265. <https://doi.org/10.1109/JSTARS.2012.2188780>
- Nickolls J, Dally WJ (2010) The GPU computing era. *IEEE Micro* 30(2):56–69. <https://doi.org/10.1109/MM.2010.41>
- NVIDIA (2015) NVIDIA Tegra X1. <http://www.nvidia.com/object/tegra-x1-processor.html>
- NVIDIA (2016) NVIDIA Tesla P100. <https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf>
- Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. *Comput Graphics Forum* 26(1):80–113. <https://doi.org/10.1111/j.1467-8659.2007.01012.x>
- Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC (2008) GPU computing. *Proc IEEE* 96(5):879–899. <https://doi.org/10.1109/JPROC.2008.917757>
- Paul J, He J, He B (2016) GPL: a GPU-based pipelined query processing engine. In: Proceedings of the 2016 international conference on management of data (SIGMOD). <https://doi.org/10.1145/2958459.2958500>

- MOD'16). ACM, New York, pp 1935–1950. <https://doi.org/10.1145/2882903.2915224>
- Qualcomm (2017) Snapdragon 835 Mobile Platform. <https://www.qualcomm.com/products/snapdragon/processors/835>
- Shimokawabe T, Aoki T, Muroi C, Ishida J, Kawano K, Endo T, Nukada A, Maruyama N, Matsuoka S (2010) An 80-fold speedup, 15.0 tflops full GPU acceleration of non-hydrostatic weather model asua production code. In: Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis (SC'10). IEEE computer society, Washington, DC, pp 1–11. <https://doi.org/10.1109/SC.2010.9>
- Wikipedia (2011) Semiconductor package. https://en.wikipedia.org/wiki/Semiconductor_package
- Wikipedia (2017) Apple mobile application processors. https://en.wikipedia.org/wiki/Apple_mobile_application_processors?oldid=752966303

Grammar-Based Compression

Sebastian Maneth

Department of Mathematics and Informatics,
Universität Bremen, Bremen, Germany

Synonyms

Dictionary-based compression; Graph grammars;
Straight-line context-free tree grammars;
Straight-line programs

Definitions

Grammar-based compression means to represent an object by a grammar that generates the object. For instance, a string can be represented by a context-free grammar that only generates the string, or, a node-labeled tree can be represented by a context-free tree grammar that generates only the tree. Two main questions are how to construct a small grammar for a given object and how to execute algorithms directly over grammars (without decompression).

Overview

This entry presents grammar-based compression of three types of finite objects: strings, trees, and (hyper)graphs, using context-free string, tree,

and hyperedge-replacement grammars. The entry (Bannai 2016) focuses on grammar construction algorithms for strings. Here, only a few grammar construction algorithms are discussed. As a prototypical example of algorithms over grammars, the equivalence problem is considered.

Key Research Findings

String Grammars

A string s consists of a sequence $a_1 \cdots a_n$ of letters a_i of an alphabet Σ . The length n of string $s = a_1 \cdots a_n$ is denoted by $|s|$.

Definition 1 Let Σ be an alphabet. A *straight-line program* (over Σ), for short *SLP*, is a triple $P = (\mathcal{N}, S, \mathcal{R})$ where \mathcal{N} is a finite set of *nonterminals* (disjoint from Σ), $S \in \mathcal{N}$ is the *start nonterminal*, and \mathcal{R} is a mapping from \mathcal{N} to (non-empty) strings over $\mathcal{N} \cup \Sigma$ such that the binary relation $\tau = \{(A, B) \in \mathcal{N}^2 \mid B \text{ occurs in } R(A)\}$ defines a single directed (ordered) acyclic graph (DAG) rooted at S .

The condition concerning τ will be referred to as “the DAG condition.” The DAG condition guarantees that the SLP P represents exactly one string $\text{val}(P)$ and that P does not contain unreachable (useless) nonterminals. The string $\text{val}(P)$ can be obtained from the string S by repeatedly replacing nonterminals X by their *rule* $R(X)$. For instance, let $n \geq 1$ and let P_n be the SLP with the rules

$$\begin{aligned} R(S) &= A_1 A_1, \\ R(A_i) &= A_{i+1} A_{i+1} \text{ for } 1 \leq i \leq n-1 \\ R(A_n) &= ab. \end{aligned}$$

Then $\text{val}(P_n)$ is the string $abab \cdots ab$ with 2^n occurrences of ab . The *size* of an SLP P is $\sum_{X \in \mathcal{N}} |R(X)|$. Note that the size of P_n is $2(n+1)$ and that $|\text{val}(P_n)| = 2^{n+1}$. Thus, an SLP can be exponentially smaller than the string it represents. At the same time, $|\text{val}(P)| \leq 3^{1+m/3}$ for any SLP P with $m = |P|$. This is because for a set of integers $I = \{|R(X)| \mid X \in \mathcal{N}\}$ with

$\sum_{i \in I} i \leq m$, it holds that $\prod_{i \in I} \leq 3^{\lceil m/3 \rceil}$. The origins of grammar-based compression are often attributed to Nevill-Manning and Witten (1997) and Kieffer and Yang (2000) and Kieffer et al. (2000); however as pointed out in Casel et al. (2016), the concept appears already in Storer and Szymanski (1982, 1978) as a particular version of their “external pointer macro.”

Grammar Compressors

A (string) grammar compressor is an algorithm that takes as input a string s and computes a small SLP P with $\text{val}(P) = s$. Several known compression schemes (e.g., LZ78 and with a small blowup also LZ77 (Rytter 2003)) can be seen as grammar compressors. Determining for a given string s and an integer k , whether there exists an SLP P with $|P| \leq k$, is NP-complete (Charikar et al. 2005). In fact, even for “1-level SLPs”, i.e., SLPs where only the start rule contains nonterminals, the problem remains NP-complete (even for an alphabet of size 5) (Casel et al. 2016); the same paper also presents an algorithm which uses dynamic programming to construct a smallest grammar in time $O(3^{|s|})$.

RePair (Larsson and Moffat 1999) is an example of a grammar compressor that runs in linear time and performs well in practice. The idea of RePair is to iteratively replace repeating diagrams (= two consecutive symbols) by new nonterminals (with their corresponding rules), until no repeating diagram exists. The *approximation ratio* of RePair, i.e., the maximum for any string s of length n of the size of an SLP produced by RePair for s divided by the size of a smallest grammar for s , is bounded by $O((n/\log n)^{2/3})$. Another well-known grammar compressor is *Bisection* (Kieffer et al. 2000), with an approximation ratio bounded by $O((n/\log n)^{1/2})$. The idea is to recursively cut the string s into two pieces, starting at position 2^j , where j is the largest integer such that $2^j < |s|$. Afterward, generate a new nonterminal for every distinct string of length greater than one produced in the cutting process. Then $|R(X)| = 2$ for every nonterminal. A space optimal online grammar compressor is presented in Takabatake et al. (2017); see Bannai (2016) for a more

detailed discussion on approximation algorithms of a smallest grammar. It should be noted that the length of the *LZ77 factorization* of a string s is a lower bound to the size of any SLP for s (Rytter 2003; Charikar et al. 2005).

Random Access

Given an SLP P one can construct in linear time (on a RAM) a data structure which supports random access to any given position p of $\text{val}(P)$ in time $\log(N)$, where $N = |\text{val}(P)|$; the substring starting at p of length m can be computed in additional $O(m)$ time (Bille et al. 2015). Moreover, for a string representing the traversal string of a tree (consisting of opening and closing parenthesis, plus (possibly) node labels), they construct in linear time (on a RAM) a data structure that supports a wide range of tree navigation operations (such as child, parent, level ancestor, lowest common ancestor, etc.).

An algorithm that runs in polynomial time with respect to $|P|$ is called *speed-up algorithm*. Many speed-up algorithms are known, for instance, for pattern matching, regular expression matching, edit-distance computation (Hermelin et al. 2009; Takabatake et al. 2016), and equality checking; see (Lohrey 2012) for a survey.

Equality Checking

Equality checking, i.e., deciding if $\text{val}(P_1) = \text{val}(P_2)$ for SLPs P_1, P_2 , can be achieved in time $O((|P_1| + |P_2|)^2)$. This can be shown by *recompression* (Jez 2015). Recompression is the application of two operations in alternation, (1) block compression and (2) pair compression. Block compression means to represent a string s of the form $a_1^{n_1} a_2^{n_2} \dots a_k^{n_k}$ by the string $a_1^{(n_1)} \dots a_k^{(n_k)}$, where the $a_i^{(n_i)}$ are new symbols. Given a partition Σ_l, Σ_r of the alphabet Σ , pair compression means to replace each digram ab with $a \in \Sigma_l$ and $b \in \Sigma_r$ by the new symbol $\langle ab \rangle$. It can be shown that for a string s that does not contain any factor of the form aa for $a \in \Sigma$, a partition Σ_l, Σ_r can be computed so that pair compression generates a string of length $\leq \frac{1}{4} + \frac{3}{4}|s|$. Two strings are equal if and only if their block or pair compressed versions are

equal. Thus, equality of two strings s_1, s_2 can be tested by repeatedly applying the three operations, (i) block compression, (ii) finding a partition, and (iii) pair compression, until one string of length one is obtained. At most $O(\log |s_1|)$ repetitions are needed. It can be shown that the three operations can be efficiently applied to SLPs P_1, P_2 directly. For instance, to carry out block compression over an SLP, one first computes for each nonterminal X the symbols a, b and maximal numbers i, j such that the string generated by X equals $a^i \alpha b^j$ and then uses this information to construct a block compressed SLP. The finally obtained SLPs are isomorphic (which can be checked in linear time) if and only if $\text{val}(P_1) = \text{val}(P_2)$. This yields an algorithm for equality checking of SLPs that runs in time $O((|P_1| + |P_2|)^2)$.

Tree Grammars

Let Σ be an alphabet. A (*node-labeled, ordered*) tree (over Σ) can be represented by a string $t = \sigma(t_1, \dots, t_n)$, where $n \geq 0$, $\sigma \in \Sigma$ and t_i is a tree. The string t_i represents the i -th subtree of the σ -labeled root node of t . The size of a tree t is its number of nodes (i.e., the number of Σ -symbols in t). If Σ is given with a mapping $\text{rk} : \Sigma \rightarrow \{0, 1, \dots\}$, then it is a *ranked alphabet*. A ranked tree (over Σ, rk) is a string $\sigma(t_1, \dots, t_n)$, where $\sigma \in \Sigma$, $n = \text{rk}(\sigma)$, and t_i is a ranked tree. For a ranked tree, parentheses are not needed, i.e., the tree $t = \sigma(t_1, \dots, t_n)$ is uniquely represented by its depth-first left-to-right string $\text{dflr}(t) = \sigma \text{dflr}(t_1) \dots \text{dflr}(t_n)$.

Since trees are strings, they can be compressed using SLPs (as, e.g., in Bille et al. 2015). However, for carrying out common tree operations, such as testing membership in a regular tree language, it is more convenient to use *tree grammars* instead of SLPs. A *straight-line regular tree grammar* (SLRT) is an SLP where each rule $R(X)$ is a tree in which nonterminals only appear at leaves. It is easy to see that SLRTs are just a notation for DAGs; thus the minimal SLRT for a tree t is unique and isomorphic to the minimal DAG of t . The minimal DAG of a tree can be computed in linear time (Downey et al. 1980).

For simplicity, only ranked trees are considered for the rest of this section. While minimal SLRTs can give exponential compression (viz., a full binary tree), they are not able to compress a monadic tree of the form $a(a(\dots a(e) \dots))$. A *straight-line context-free tree grammar* (over Σ) (SLCT) is a triple (\mathcal{N}, S, R) , where \mathcal{N} is a ranked alphabet of nonterminals, $S \in \mathcal{N}$ is the start nonterminal, and a rule $R(X)$ is a ranked tree over $\mathcal{N} \cup \Sigma$ and parameters y_1, \dots, y_m of rank zero. It is required that the DAG condition (see Definition 1 and the sentence below it) holds. A rule $R(X)$ is applied to a subtree $X(t_1, \dots, t_m)$ by replacing the subtree by the tree obtained from $R(X)$ by replacing each y_j by t_j for $1 \leq j \leq m$.

Note that SLCTs can achieve double-exponential compression: let T_n be the SLCT with

$$\begin{aligned} R(S) &= A_1(A_1(e)), \\ R(A_i) &= A_{i+1}(A_{i+1}(y_1)), 1 \leq i \leq n-1, \\ R(A_n) &= f(y_1, y_1). \end{aligned}$$

Then $\text{val}(T_n)$ is a full binary tree of height $2^n + 1$ and size $2^{2^n+1} - 1$.

A *tree straight-line program* (TSLP) is a SLCT such that each y_j occurs exactly once in $R(X)$ for any nonterminal X . TSLPs can achieve only single exponential compression and are better behaved than SLCTs. The rest of this section only deals with TSLPs. In the literature, TSLPs are often called “linear straight-line context-free tree grammars.”

Grammar Compressors

RePair compression can be generalized to construct a small TSLP for a given tree (Lohrey et al. 2013). A digram is now a triple (f, i, g) where f, g are node labels and $1 \leq i \leq \text{rk}(f)$. For instance, the tree $f(f(a, g), g)$ has two occurrences of the digram $(f, 2, g)$. A digram is replaced by a nonterminal A of rank $\text{rk}(f) + \text{rk}(g)$, and an appropriate rule for A is introduced; here $R(A) = f(y_1, g)$. The TreeRePair algorithm is controlled by a maximal rank parameter m to produce TSLPs where the rank of each nonterminal is $\leq m$. This is because many algorithms require time proportional to the

rank of nonterminals; hence it is desirable to keep this number small. TreeRePair works well in practice, but from a theoretical point of view does not approximate well a smallest grammar. For a linear time approximation algorithm that generalizes the recompression idea of SLPs to TSLPs, it has been shown that for a tree t of size n , a TSLP G of size $O(rg + rg \log(n))$ is produced, where r is the maximal rank of a node label in t and g is the size of a smallest grammar (Jez and Lohrey 2016). For another compressor which generalizes Bisection to trees, it was shown that a TSLP of size $O(n \log_\sigma n)$ is produced, where n is the size of the input tree and σ is the number of node labels (Ganardi et al. 2017).

Equality Checking

Similarly as for SLPs, many speed-up algorithms are known for TSLPs, for instance, testing membership in a regular tree language or testing equality. Often the complexity of algorithms of TSLPs depends on the maximal number of parameters of a nonterminal. It has been shown that any given TSLP can be transformed into an equivalent one where each nonterminal uses at most one parameter. For such a TSLP each rule has one of four forms

- (1) $f(A_1, \dots, A_n),$
- (2) $B(C),$
- (3) $f(A_1, \dots, A_i, y_1, A_{i+1}, \dots, A_n),$
- (4) $B(C(y_1)).$

Testing equality of TSLPs T_1, T_2 can easily be reduced to the string case by constructing SLPs P_1, P_2 for the strings $\text{dflr}(\text{val}(T_i))$. The size of P_i is $O(m|T_i|)$, where m is the maximal rank of a nonterminal of P_i . TSLPs can also be used to compress *unordered trees*, by considering the tree $\text{val}(G)$ as unordered. Isomorphism of unordered trees represented by TSLPs T_i can be solved in polynomial time (Lohrey et al. 2015), by computing TSLPs T'_i with $\text{val}(T'_i) = \text{canon}(\text{val}(T_i))$. The canon of a tree t is obtained from t by ordering the subtrees of each node length-lexicographically

with respect to their traversal string (according to some fixed order on Σ). For instance, $\text{canon}(f(d(a), c(a), b)) = f(b, c(a), d(a))$, if c is smaller than d . The result is extended to unrooted unordered trees by constructing TSLPs for rooted version of the trees of the given TSLPs, rooted at the (at most two) center nodes of the trees.

Graph Grammars

There are two well-known notions of context-free graph grammars: node-replacement (NR) grammars and hyperedge-replacement (HR) grammars (Engelfriet 1997). Compression via straight-line HR grammars has been considered recently (Maneth and Peternek 2017).

Let Σ be an alphabet and Γ a ranked alphabet with $\text{rk}(\gamma) \geq 1$ for every $\gamma \in \Gamma$. A (directed hyper) graph over (Σ, Γ) is a tuple $(V, E, \lambda, \text{att}, \text{ext})$ where V and E are finite sets of nodes and (hyper) edges, λ maps each $v \in V$ to Σ and each $e \in E$ to Γ . The attachment mapping att maps each $e \in E$ to a string over V of length $n = \text{rk}(\lambda(e))$. The string $\text{ext} \in V^*$ defines the sequence of *external nodes*. Typically, hypergraphs we compress do not have external nodes (i.e., ext equals the empty string ε); however, in a graph g appearing in the right-hand side of a grammar rule, external nodes indicate how g is merged with the host graph (see below). The graph is of *rank* k if $k = |\text{ext}|$. A conventional (node and edge labeled) graph is a hypergraph where (i) each $\gamma \in \Gamma$ has rank 2, (ii) if $\text{att}(e) = v_1 v_2$ then edge e is directed from v_1 to v_2 , and (iii) $\text{ext} = \varepsilon$.

A *straight-line hyperedge-replacement grammar* (over (Σ, Γ)) (SLHR) is a triple (\mathcal{N}, S, R) , where \mathcal{N} is a ranked alphabet of nonterminals; $S \in \mathcal{N}$ is the start nonterminal, for every $X \in \mathcal{N}$; and the rule $R(X)$ is a hypergraph over $(\Sigma, \mathcal{N} \cup \Gamma)$ of rank $\text{rk}(X)$, and the DAG condition (see Definition 1 and the sentence below it) holds. Let e be an X -labeled edge with $\text{att}(e) = v_1 \dots v_k$. Then a rule $R(X)$ is applied to e by removing e , inserting a disjoint copy of $R(X)$, and then identifying the i -th external node of the copy of $R(X)$ with the node v_i . The external nodes of an

HR grammar have a similar role as the parameters of a context-free tree grammar.

Let $g = (V, E, \lambda, \text{att}, \text{ext})$ be a graph. The *node size* of g is $|V|$. The *edge size* of g is $|\{e \in E \mid \text{rk}(\lambda(e)) \leq 2\}| + \sum\{\text{rk}(\lambda(e)) \mid \text{rk}(\lambda(e)) \geq 3\}|$. The *size* of g is the sum of node and edge sizes. The size of an SLHR grammar is $\sum_{X \in \mathcal{N}} |R(X)|$.

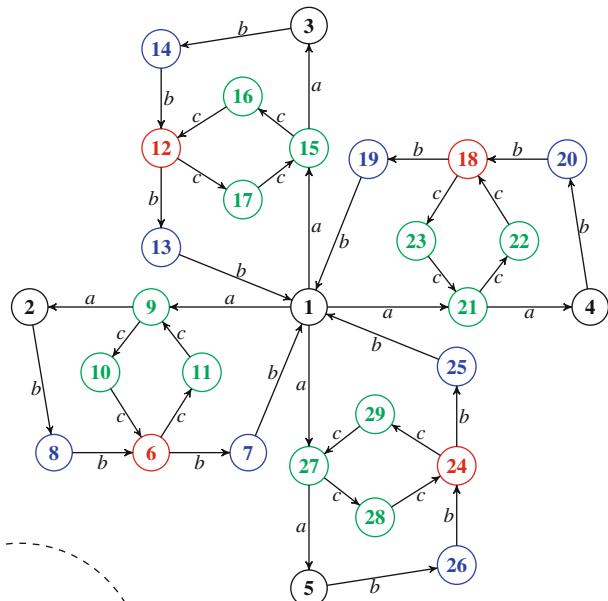
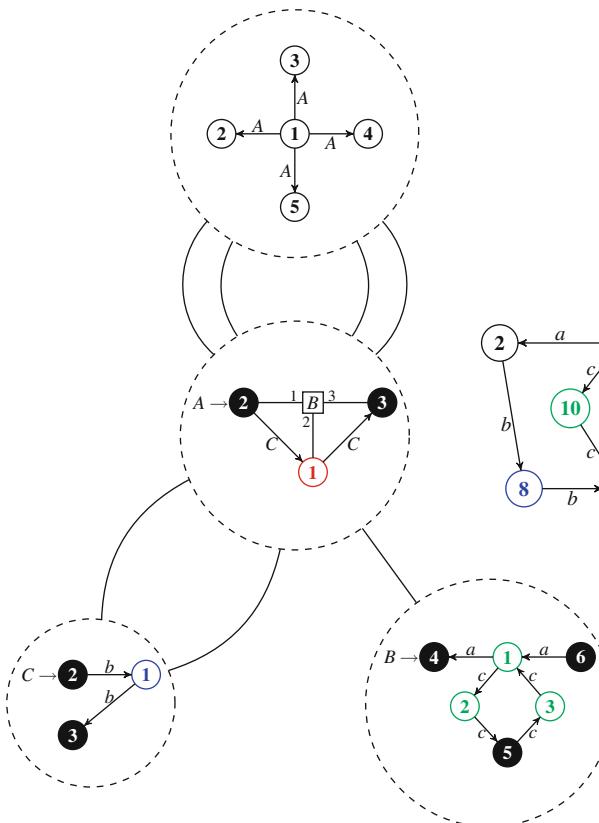
Consider the graph g in right of Fig. 1. This graph is edge-labeled and has no node labels. The nodes are ordered according to the numbers shown in the nodes. The size of the graph is 69. The left of Fig. 1 shows the “derivation DAG” (cf. the DAG condition of Definition 1) of an SLHR grammar G representing g . A rule $R(X) = h$ is written $X \rightarrow h$ and simply h in case of the start nonterminal S . The size of G is 34 (17 nodes, 14 edges, and 1 hyperedge of size 3). Note that there is a smaller grammar, obtained by removing

the nodes 2 to 5 from the start graph (and using A -hyperedges of rank 1). The external nodes are colored black with their order according to the node numbers. Clearly, $\text{val}(G) = g$. The node numbers can be obtained by following a specific order of the nonterminal edges in each rule, namely, lexicographically w.r.t the sequence of incident nodes. The node numbers of nodes 1 to 5 in $R(S)$ stay as they are. The A -edge from node 1 to node 2 is derived first. This generates a new red node labeled 6 (the first free label). Next is the C -edge from 1 to 6 which creates the blue node 8. Next, the green nodes 9, 10, and 11 are generated by application of the B -rule.

G

Grammar Compressors

RePair compression can be generalized to hypergraphs by defining a diagram to be a graph



Grammar-Based Compression, Fig. 1 Derivation DAG of an SLHR grammar G together with the graph $\text{val}(G)$ represented by G

g consisting of two edges plus their incident nodes. Further, the two edges in g must have at least one common incident node, and g must contain $k \geq 1$ distinct external nodes. The RePair idea is to (repeatedly) replace in a graph all occurrences of a most frequent digram by edges labeled by a new nonterminal. However, an issue that is straightforwardly solved in strings and trees turns out to be problematic in the graph case: when counting digram occurrences, only *non-overlapping* occurrences of digrams are of interest. For instance, the string *aaaaaa* contains *four* occurrences of the digram *aa*, but the number of non-overlapping occurrences of *aa* is *two*. In strings (trees) a maximal set of non-overlapping occurrences of a digram can be determined by simply counting greedily in a left-to-right (bottom-up) traversal. Note that in strings (trees), digrams can only overlap if the two letters (node labels) are equal. This is not the case for graphs. The fastest known method for finding a maximal set of non-overlapping digrams in a graph g requires $O(|g|^4)$ time. The RePair algorithm for graphs therefore resorts to a linear time approximation which greedily counts digrams according to a given node order. Interestingly, best results are achieved for a node order where “similar nodes” are visited in order. The order is inspired by the numbering scheme used in the Weisfeiler-Lehman isomorphism test (Weisfeiler and Lehman 1968).

Note that a TSLP P can be seen as a particular SLHR: a node with k children in the right-hand side of P is considered as a hyperedge incident with $k + 1$ nodes (the first node is incident to the edge of the parent node, the rest are incident to the edges of the child nodes). An SLCT (i.e., a TSLP in which parameters may be copied) can also be seen as an SLHR, however, as one that generates a DAG, not a tree. One may wonder if a tree-generating SLHR grammar can achieve better compression than a TSLP. This is not the case (apart from a constant factor): for any tree-generating SLHR G , one can construct an equivalent SLHR G' so that $\text{val}(G') = \text{val}(G)$, $|G'| \leq 2|G|$, and each right-hand side of G' is a tree.

Examples of Application

Grammar-based compression is a mathematical abstraction of known compression schemes, such as LZ78 (Ziv and Lempel 1978) for strings or common subexpression sharing via DAGs for trees (see, e.g., Ershov 1958). While the known schemes are extensively used within applications from industry, there is few mention of “grammar-based compression” within industrial applications. An example of an exception is (Senin et al. 2015) where grammar-based compression is applied to detect anomalies in time series; for this purpose they implemented the tool “GrammarViz” (Senin et al. 2014). Particular variants of grammar-based compression are applied to infer structure from RNA (Zhao et al. 2015; Liu et al. 2008). In Tabei et al. (2016), an algorithm and implementation is presented that performs partial least squares regression (PLS) over data matrices that are grammar-compressed; their results show great improvement in working space and scalability, over uncompressed PLS computations. Grammar-based compression is regularly mentioned in the context of XML compression (see, e.g., Sakr 2009); an implementation of an XPath query evaluator operating directly over TSLPs is presented in Maneth and Sebastian (2010).

Future Directions of Research

In terms of tree grammars, no compressor has yet been constructed for *non-linear* straight-line context-free tree grammars (as mentioned, such grammars allow double-exponential compression). In fact, also with respect to theoretical results, e.g., concerning the availability of speed-up algorithms for such grammars, almost nothing is known yet.

For SLHR grammars G , only a few speed-up algorithms have been investigated so far (Maneth and Peternek 2017). For instance, given two node numbers u, v , it can be determined in time $O(|G|)$ whether or not v is reachable from u .

Given a regular expression α and two node numbers u, v , it can be determined in time $O(|\alpha||G|)$ whether there is a path from u to v that matches α . The existence of such nodes can be determined in the same time bound. Many problems concerning SLHR grammars are still open. For instance, for a given k , it is not known whether isomorphism of two graphs given by SLHRs in which each right-hand side has size $\leq k$ can be solved in polynomial time. The corresponding graphs are of bounded tree width, and for uncompressed such graph isomorphism is decidable in polynomial time (Bodlaender 1990).

Cross-References

- ▶ [Graph Data Models](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [RDF Compression](#)

References

- Bannai H (2016) Grammar compression. In: Encyclopedia of Algorithms, Springer, pp 861–866. https://doi.org/10.1007/978-1-4939-2864-4_635
- Bille P, Landau GM, Raman R, Sadakane K, Satti SR, Weimann O (2015) Random access to grammar-compressed strings and trees. *SIAM J Comput* 44(3):513–539. <https://doi.org/10.1137/130936889>
- Bodlaender HL (1990) Polynomial algorithms for graph isomorphism and chromatic index on partial k -trees. *J Algorithms* 11(4):631–643. [https://doi.org/10.1016/0196-6774\(90\)90013-5](https://doi.org/10.1016/0196-6774(90)90013-5)
- Casel K, Fernau H, Gaspers S, Gras B, Schmid ML (2016) On the complexity of grammar-based compression over fixed alphabets. In: Proceeding of 43rd international colloquium on automata, languages, and programming, ICALP 2016, 11–15 July 2016, Rome, pp 122:1–122:14. <https://doi.org/10.4230/LIPIcs.ICALP.2016.122>
- Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Sahai A, Shelat A (2005) The smallest grammar problem. *IEEE Trans Information Theory* 51(7):2554–2576. <https://doi.org/10.1109/TIT.2005.850116>
- Downey PJ, Sethi R, Tarjan RE (1980) Variations on the common subexpression problem. *J ACM* 27(4):758–771. <http://doi.acm.org/10.1145/322217.322228>
- Engelfriet J (1997) Context-free graph grammars. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages: beyond words, vol 3. Springer, Berlin/Heidelberg, pp 125–213. https://doi.org/10.1007/978-3-642-59126-6_3
- Ershov AP (1958) On programming of arithmetic operations. *Commun ACM* 1(8):3–9
- Ganardi M, Hucke D, Jez A, Lohrey M, Noeth E (2017) Constructing small tree grammars and small circuits for formulas. *J Comput Syst Sci* 86:136–158. <https://doi.org/10.1016/j.jcss.2016.12.007>
- Hermelin D, Landau GM, Landau S, Weimann O (2009) A unified algorithm for accelerating edit-distance computation via text-compression. In: Proceedings of the 26th international symposium on theoretical aspects of computer science, STACS 2009, 26–28 Feb 2009, Freiburg, pp 529–540. <https://doi.org/10.4230/LIPIcs.STACS.2009.1804>
- Jez A (2015) Faster fully compressed pattern matching by recompression. *ACM Trans Algorithms* 11(3):20:1–20:43. <http://doi.acm.org/10.1145/2631920>
- Jez A, Lohrey M (2016) Approximation of smallest linear tree grammar. *Inf Comput* 251:215–251. <https://doi.org/10.1016/j.ic.2016.09.007>
- Kieffer JC, Yang E (2000) Grammar-based codes: a new class of universal lossless source codes. *IEEE Trans Inf Theory* 46(3):737–754. <https://doi.org/10.1109/18.841160>
- Kieffer JC, Yang E, Nelson GJ, Cosman PC (2000) Universal lossless compression via multilevel pattern matching. *IEEE Trans Inf Theory* 46(4):1227–1245. <https://doi.org/10.1109/18.850665>
- Larsson NJ, Moffat A (1999) Offline dictionary-based compression. In: Data Compression Conference, DCC 1999, Snowbird, 29–31 Mar 1999, pp 296–305. <https://doi.org/10.1109/DCC.1999.755679>
- Liu Q, Yang Y, Chen C, Bu J, Zhang Y, Ye X (2008) Rnacompress: grammar-based compression and informational complexity measurement of rna secondary structure. *BMC Bioinform* 9(1):176. <https://doi.org/10.1186/1471-2105-9-176>
- Lohrey M (2012) Algorithmics on SLP-compressed strings: a survey. *Groups Complex Cryptol* 4(2):241–299. <https://doi.org/10.1515/gcc-2012-0016>
- Lohrey M, Maneth S, Mennicke R (2013) XML tree structure compression using RePair. *Inf Syst* 38(8):1150–1167. <https://doi.org/10.1016/j.is.2013.06.006>
- Lohrey M, Maneth S, Peternek F (2015) Compressed tree canonization. In: Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015, Part II, Kyoto, 6–10 July 2015, pp 337–349. https://doi.org/10.1007/978-3-662-47666-6_27
- Maneth S, Peternek F (2017) Grammar-based graph compression. *CoRR* abs/1704.05254. <http://arxiv.org/abs/1704.05254>, 1704.05254
- Maneth S, Sebastian T (2010) Fast and tiny structural self-indexes for XML. *CoRR* abs/1012.5696. <http://arxiv.org/abs/1012.5696>, 1012.5696
- Nevill-Manning CG, Witten IH (1997) Identifying hierarchical structure in sequences: a linear-time algorithm. *J Artif Intell Res* 7:67–82. <https://doi.org/10.1613/jair.374>

- Rytter W (2003) Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor Comput Sci* 302(1–3):211–222. [https://doi.org/10.1016/S0304-3975\(02\)00777-6](https://doi.org/10.1016/S0304-3975(02)00777-6)
- Sakr S (2009) XML compression techniques: a survey and comparison. *J Comput Syst Sci* 75(5):303–322. <https://doi.org/10.1016/j.jcss.2009.01.004>
- Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S, Lerner M (2014) Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In: Proceedings of the European conference on machine learning and knowledge discovery in databases – , ECML PKDD 2014, Nancy, Part III, 15–19 Sept 2014, pp 468–472. https://doi.org/10.1007/978-3-662-44845-8_37
- Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S (2015) Time series anomaly discovery with grammar-based compression. In: Proceedings of the 18th international conference on extending database technology, EDBT 2015, Brussels, Belgium, 23–27 March 2015, pp 481–492. <https://doi.org/10.5441/002/edbt.2015.42>
- Storer JA, Szymanski TG (1978) The macro model for data compression (extended abstract). In: Proceedings of the 10th annual ACM symposium on theory of computing, 1–3 May 1978, San Diego, pp 30–39. <http://doi.acm.org/10.1145/800133.804329>
- Storer JA, Szymanski TG (1982) Data compression via textural substitution. *J ACM* 29(4):928–951. <http://doi.acm.org/10.1145/322344.322346>
- Tabei Y, Saigo H, Yamanishi Y, Puglisi SJ (2016) Scalable partial least squares regression on grammar-compressed data matrices. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, 13–17 Aug 2016, pp 1875–1884. <http://doi.acm.org/10.1145/2939672.2939864>
- Takabatake Y, Nakashima K, Kuboyama T, Tabei Y, Sakamoto H (2016) SIEDM: an efficient string index and search algorithm for edit distance with moves. *Algorithms* 9(2):26. <https://doi.org/10.3390/a9020026>
- Takabatake Y, Tomohiro I, Sakamoto H (2017) A space-optimal grammar compression. In: Proceedings of 25th annual European symposium on algorithms, ESA 2017, 4–6 Sept 2017, Vienna, pp 67:1–67:15. <https://doi.org/10.4230/LIPIcs.ESA.2017.67>
- Weisfeiler B, Lehman AA (1968) A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, Seriya 2*(9):12–16 (in Russian)
- Zhao Y, Hayashida M, Cao Y, Hwang J, Akutsu T (2015) Grammar-based compression approach to extraction of common rules among multiple trees of glycans and rnas. *BMC Bioinform* 16(1):128. <https://doi.org/10.1186/s12859-015-0558-4>
- Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. *IEEE Trans Information Theory* 24(5):530–536. <https://doi.org/10.1109/TIT.1978.1055934>

Graph Aggregation

- Graph OLAP

Graph Analytics

- Graph OLAP

Graph Benchmarking

Khaled Ammar

Thomson Reuters Labs, Thomson Reuters,
Waterloo, ON, Canada

Introduction

Graphs have attracted research efforts for about three centuries (Biggs et al. 1986). Earlier, the focus was mainly on mathematical models and algorithms. Graph theory continues to be an active research area that includes many open problems. In the last decade, there has been a considerable amount of research in analyzing and processing large graph structures for real applications. This effort has led to the development of several alternative algorithms, techniques, and big data systems.

Parallel systems are very important and big data challenges are relevant to graph analytics. Although the edge list of some existing graphs are relatively small in size and may fit in one machine, these graphs are often large when considering vertex properties, graph indexes, and intermediate results during query processing. Processing often leads to very large memory requirements, which makes it not practical to fit the entire graph and its data into a single machine.

Big graph data has a special property that makes it harder than typical big data; there is no locality. For example, an employee database could be partitioned based on the employee department. Executing queries and creating reports about each department does not need interaction

between partitions (departments). Therefore, it is possible to keep each department in a machine to parallelize the workload. On the other hand, graph processing is interested in the relationships between entities and queries the relationship structure rather than individual entities. Therefore, when considering the relationship graph of employees, partitions (departments) are expected to be queried together.

The lack of locality in graph data and the need of running an algorithm for multiple iterations before convergence make data shuffling between partitions the major cost. Furthermore, due to the power-law property of many real graphs, each partition/machine needs to make different amount of workload. Most parallel systems try to minimize the data shuffling cost while maintaining balanced workload. They take different approaches to do that. Therefore benchmarking these systems should take into consideration several aspects, such as, memory usage, CPU utilization, graph data, and so on.

This entry briefly introduces the readers to graph data, real graph properties, and synthetic graph generation. Then, it discusses common workloads for graph benchmarking. After that, it summarizes existing benchmarks. The last section discusses experiment design, metrics to be measured, and evaluating the parallel cost.

Graph Data

A graph, G , is defined as $G = (V, E)$, where V is a set of vertices and E is a set of edges, such that $|V| = N, |E| = M$. For an edge $e = (u, v, P_e)$, $e \in E$ if $v, u \in V$ and P_e is a set of properties associated with edge e . A vertex $v \in V$, also, has a set of properties, P_v . P_e and P_v can be empty. If $P_v \neq \emptyset$ or $P_e \neq \emptyset$, it is known as a *property graph*. This definition is flexible to accommodate several graph types, such as directed and undirected, weighted edges, and bipartite graphs among others.

Type of graph used depends on the semantics of the application in hand. The web graph is a directed graph because each edge represents a hyper link from one web page to the other.

Some social networks, such as Facebook, adopt the “friendship” relationship; hence, the friends graph in Facebook is undirected. However, Twitter adopts the “following” relationship; hence, it is represented as a directed graph. A recommendation system graph, between customers and products, is a bipartite graph, where $V = V_1 \cup V_2$ such that V_1 represents customers, V_2 represents products, and edges between the two sets represent preferences.

There are several sources of real graph data in the literature, such as SNAP (Leskovec and Krevl 2014) and LWA (Boldi et al. 2011). Some projects are dedicated for preparing large real graphs, for example, Common Crawl has 64 billion edges. These real datasets are complementary to synthetic datasets. Synthetic datasets are equally important because they allow a benchmark to examine different properties of the same dataset. The main challenge in generating a synthetic dataset is to ensure that it represents the real world. The remaining of this section will discuss the properties of real graphs and synthetic graph generators.

Real Graphs Properties

The problem of studying the patterns and properties of real web and social graphs have been addressed extensively in the literature (Leskovec et al. 2005b; Faloutsos et al. 1999; Chakrabarti et al. 2010). Table 1 shows a few real graph datasets (Boldi et al. 2004, 2011; Boldi and Vigna 2004; Leskovec and Krevl 2014) and their properties. These properties include graph size, average and maximum degrees, diameter, and the percentage of nodes included in the strong connected component.

Social networks and web graphs are very large and have a high growth rate in terms of vertices and edges. They are sparse; the average vertex degree is significantly smaller than the number of vertices; however, the graph density increases as the graph grows. The diameter, which is the longest shortest path between any two vertices in the graph, is typically small and shrinking as the graph grows. Most of these graphs have one very large connected component and a few very small components that are usually ignored. The vertex

Graph Benchmarking, Table 1 Properties of some real graph datasets

Domain	Dataset	$ E $	Avg./Max. degree	Diameter	SCC
Social network	soc-LiveJournal	68 M	14.2/15 K	16	100%
	com-Orkut	117 M	38.1/33 K	9	100%
	com-Friendster	1.8 B	27.5/4223	32	100%
Road network	California	2.7 M	1.4/12	849	99.6%
	Texas	1.9 M	1.4/12	1054	97.9%
Citation graph	US patent	16 M	4.4/779	22	100%
Web graph	uk-2007-05	3.7 B	35.3/975 K	22.78	64.7%
	clueweb12	42.5 B	43.5/75 M	15.7	79%

degree follows the power-law distribution with a very long tail.

Existing graph generators try to represent web and social graph properties. There are typically seven properties:

- Very large
- Sparse
- Small diameter
- One large connected component
- Community structure
- Power law degree distribution
- Dynamic

It is important to acknowledge that these are not the properties of all real graphs. Graphs have different properties based on their source. Indeed, road network graphs have different characteristics than social networks, web, and citation graphs. The average degree is smaller than that of the others; moreover, the maximum degree is very close to the average degree. Apparently, road network graphs do not follow power-law distribution as do many other graphs. Furthermore, the diameter of the graph is significantly larger than all other graphs' diameter; hence, the small diameter phenomenon is not prevalent in this domain.

Synthetic Graph Generation

Erdös-Renyi

ER (Erdös and Rényi 1960) generation model can generate random graphs such that any possible graph with N vertices and M edges have the

same probability. These random graphs could be large, sparse.

Watts-Strogatz

The WS (Watts and Strogatz 1998) model tries to add small world (short diameter) property and high clustering (one large component) to the generated random graphs. Instead of creating edges randomly as in the ER model, it has two parameters β and K , such that:

$$M = \frac{NK}{2}$$

$$N > K > \ln(N) > 1$$

$$0 \leq \beta \leq 1.$$

The model starts by connecting two vertices v_i and v_j if

$$0 < |i - j| \bmod \left(N - 1 - \frac{K}{2} \right) < \frac{K}{2}.$$

Then, for each edge (v_i, v_j) such that $i < j$, the vertex v_j is replaced by v_z with a probability β . v_z is chosen uniformly from all available vertices as long as there is no self-edge and no duplicate edges between vertices. If $\beta = 1$, then the graph is a random graph similar to ER; if $\beta = 0$, then we have a clustered graph without random edges. These random edges help to reduce the diameter.

Barabasi-Albert

The BA (Albert and Barabási 2002) model is based on the preferential attachment mechanism which means that some value is distributed to

agents based on what they already have. This mechanism can generate graphs with power-law distribution; the distributed value in this case is the number of edges.

The algorithm starts with an initial random graph G_0 with N_0 vertices. Every new vertex v added to the graph could be connected to $m < N_0$ vertices. The probability of edge (v, v_i) is $p_i = \frac{d_i}{2 \times M}$, such that d_i is the degree of vertex v_i and M is number of edges in the graph so far.

This approach can create a large sparse dynamic graph with a small diameter and power-law distribution for vertex degree, but it does not create graphs with community structure.

Kronecker Graphs

Kronecker graphs model (Leskovec et al. 2005a) was proposed as an easy to analyze mathematical model for graph generation. The model is based on the Kronecker product of an initial matrix which represents the adjacency matrix of an initial graph. The K th power of the initial matrix represents the generated graph.

The Kronecker product of two matrices A and B is matrix such that each value in position (i, j) is $a_{(i,j)} \times B$. The naive approach of fitting this model to real graphs is expensive and needs exponential time, but the authors proposed an efficient and scalable algorithm called KronFit.

RTG (Random Typing Graphs)

Miller (1957) showed that typing random characters on a keyboard with only one character identified as a separator between words can generate a list of words that follow a power-law distribution. RTG (Akoglu and Faloutsos 2009) builds on this by connecting each pair of consecutive words by an edge. The final result is a graph whose node degrees follow the power-law distribution.

Although Kronecker graph generator is able to create graphs similar to real-world graphs and enables easier analysis of the mathematical model, it has some disadvantages such as generation of multinomial/lognormal degree distributions instead of power-law and being not flexible. Waterloo Graph Benchmark (WGB) generator (Ammar

and Özsu 2014), on the other hand, is built on top of RTG but is very flexible. It is capable of generating directed/undirected, weighted/unweighted, and bipartite/unipartite/multipartite graphs.

RTG enhances the properties of the generated graphs by introducing some dynamic graph properties such as shrinking diameter and densification. Structure preference was also introduced in RTG to match the real-life graph property of having community structures, meaning that nodes form groups and possibly groups of groups. These groups are usually created by connecting similar nodes together to form a community.

G

Workloads

There are three main workload categories in graph applications: online queries, updates, and iterative queries. Online queries are read only and interactive, usually do not have high complexity, and do not require multiple computation iterations; the answer is expected to be computed quickly. Updates are also interactive, but they change the graph's structure by changing its parameters or its topology. Updates also include adding or removing a vertex or an edge. Iterative queries are heavy analytical queries, possibly requiring multiple passes over the data, and usually take a long time to complete. They may include some updates, but the changes to the graph do not need to occur in real time.

Online Queries

Many graph applications include only online queries such as a simple check of the availability of customers' relationship to a retail shop, an update for the news feed in social media, etc. Examples are:

- Find a vertex or edge by its properties
- Report K-hop neighborhood of a query
- Reachability or shortest path query between two vertices
- Pattern matching query

Iterative Queries

Iterative queries are usually analytical workload over graph datasets (graph analytics). There is a large number of analysis algorithms such as subgraph discovery, finding important nodes (PageRank), attack detection, diameter computation, topic detection, and triangle counting. There are several alternatives for each of these problems.

Existing Graph Benchmarks

HPC Scalable Graph Analysis Benchmark

The idea behind HPC Scalable Graph Analysis Benchmark (Bader and Madduri 2005) is to develop an application with multiple kernels that access a weighted directed graph. These kernels represent multiple graph queries. They require irregular access to the graph so it is not expected that a single data structure for the input graph dataset would be optimal for all kernels. There are four kernels designed to address the following operations: bulk insertion, retrieving highest weight edges, k-hops operations, and betweenness calculation. However, the benchmark does not include many other graph operations such as updates or iterative queries.

Graph Traversal Benchmark

The authors of the Graph Traversal Benchmark (Ciglan et al. 2012) classify graph data management systems in two categories: (1) graph databases such as Neo4J and (2) distributed graph processing frameworks such as Pregel (Malewicz et al. 2010) and GraphLab (Low et al. 2012). This benchmark focuses on traversal operations on graph databases. Consequently, the capability of this benchmark is limited, because it does not support graph data queries or graph analysis algorithms. Moreover, it does not consider distributed graph processing frameworks.

Graph500

Graph500 (Murphy et al. 2010) is developed by a group of experts from academia, industry, and research laboratories. The main target is evaluating supercomputing systems for data intensive

applications, focusing on graph analysis, to guide the design of hardware architectures and their software systems.

BigDataBench

BigDataBench (Wang et al. 2014) is a big data benchmark suite. It is generic and includes various types of dataset: text, graph, and table. The graph component uses Kronecker graph model to create synthetic datasets based on real datasets. In this benchmark, the data generator is developed to learn the properties of a real graph and then generate synthetic data with similar characteristics. Two real graph datasets are included: Google web graph and Facebook social graph.

Linked Data Benchmark Council

The Linked Data Benchmark Council (LDBC) has two graph benchmarks: Social Network Benchmark (SNB) (Erling et al. 2015) and GraphAnalytics (Iosup et al. 2016). SNB is a benchmark for graph databases using a large generated social network. The graph is fully connected and simulates social network users' activities during a period of time.

LDBC GraphAnalytics (Iosup et al. 2016) is a new industrial-grade benchmark for graph processing system. Many studies in the literature tried to evaluate the performance of Pregel-like open source graph processing systems (Han et al. 2014; Batarfi et al. 2015; Lu et al. 2014). LDBC GraphAnalytics also include industry-driven systems such as GraphPad (Anderson et al. 2016) from Intel and PGX (Hong et al. 2015) from Oracle. This benchmark uses real and synthetic datasets. The synthetic dataset generator is similar to the SNB benchmark.

Experiment Design

In the process of designing a benchmarking experiment, it is very important to be consistent and accurate. If the experiment includes multiple systems, a special effort should ensure that systems configuration should lead to similar resource allocation; for example, systems should be configured to use the same number of cores.

If the experiment includes multiple dimensions then evaluating the possibilities of each dimension is an independent experiment. It is important to change one dimension at a time to conclude meaningful results. Below is a list of some examples:

- Datasets
- Workloads
- Systems
- Number of cores in each machine (scale-up)
- Number of machines in the cluster (scale-out)

In the rest of this section, we will discuss two important considerations during the design process of the benchmarking experiment.

Metrics Measures

A good benchmarking experiment should measure resources utilization and system performance. High resource usage is better if the system is expected to run in isolation, while low resource usage is better if the system is expected to run in an environment such that systems share resources and run simultaneously. Nonetheless, evaluating system performance should be done in isolation, with no resource sharing across systems or across experiments.

Benchmarks should report the following measures:

- CPU utilization for each process type (user, system, I/O, idle) every second
- Memory usage every second
- Network usages (bytes and packages) before and after execution

They should also report the following components to evaluate system performance:

- Data-loading/partitioning time
- Execution time
- Result-saving time
- Total response time (latency)

Data-loading time includes reading data set from its source (e.g., HDFS) and graph partitioning,

when necessary. Ideally, total response time should equal load + execute + save. However, it should be reported separately, because it represents the end-to-end processing time, and occasionally includes some overhead that might not be explicitly reported by some systems, such as the time of repartitioning, networking, and synchronization.

Parallelization COST

Evaluating the system scalability is very important. However, parallelization usually comes at a cost. Parallel algorithms, either in multi-core or multi-machine environments, are usually more complicated. COST (McSherry et al. 2015) stands for Configuration that Outperforms a Single Thread and is used in the literature to evaluate the overhead of parallel algorithms and systems.

The main idea is that parallel algorithms are often not optimal due to the special design consideration for parallel processing between machines, such as, synchronization and communication overhead. COST factor represents the response time of one thread divided by the response time of a parallel system. The baseline should be an efficient single thread implementation, such as the single thread execution of algorithms in the GAP Suite (Beamer et al. 2015) on a machine with large memory.

Cross-References

- ▶ [Graph Generation and Benchmarks](#)
- ▶ [Metrics for Big Data Benchmarks](#)

References

- Akoglu L, Faloutsos C (2009) RTG: a recursive realistic graph generator using random typing. *Data Min Know Disc* 19(2):194–209
- Albert R, Barabási AL (2002) Statistical mechanics of complex networks. *Rev Mod Phys* 74(1):47
- Ammar K, Özsu M (2014) WGB: towards a universal graph benchmark. In: Rabl T, Raghunath N, Poess M, Bhandarkar M, Jacobsen HA, Baru C (eds) *Advancing big data benchmarks. Lecture notes in computer science*. Springer, pp 58–72. https://doi.org/10.1007/978-3-319-10596-3_6

- Anderson MJ, Sundaram N, Satish N, Patwary MMA, Willke TL, Dubey P (2016) Graphpad: optimized graph primitives for parallel and distributed platforms. In: Proceedings of the 30th international parallel and distributed processing symposium, pp 313–322
- Bader DA, Madduri K (2005) Design and implementation of the hpcg graph analysis benchmark on symmetric multiprocessors. In: International conference on high-performance computing, pp 465–476
- Batari O, Shawi R, Fayoumi A, Nouri R, Beheshti SMR, Barnawi A, Sakr S (2015) Large scale graph processing systems: survey and an experimental evaluation. *Clust Comput* 18(3):1189–1213
- Beamer S, Asanović K, Patterson D (2015) The gap benchmark suite. arXiv preprint arXiv:150803619
- Biggs N, Lloyd EK, Wilson RJ (1986) Graph theory. Clarendon Press, New York, pp 1736–1936
- Boldi P, Vigna S (2004) The webgraph framework I: compression techniques, pp 595–601
- Boldi P, Codenotti B, Santini M, Vigna S (2004) Ubicrawler: a scalable fully distributed web crawler. *Softw Pract Exp* 34(8):711–726
- Boldi P, Rosa M, Santini M, Vigna S (2011) Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Proceedings of the 20th international conference on world wide web, pp 587–596
- Chakrabarti D, Faloutsos C, McGlohon M (2010) Graph mining: laws and generators. In: Aggarwal CC (ed) Managing and mining graph data. Springer, pp 69–123
- Ciglan M, Averbuch A, Hluchy L (2012) Benchmarking traversal operations over graph databases. In: Proceedings of the workshops of 28th international conference on data engineering. IEEE, pp 186–189
- Erdős P, Rényi A (1960) On the evolution of random graphs. In: Publication of the mathematical institute of the hungarian academy of sciences, pp 17–61
- Erling O, Averbuch A, Larriba-Pey J, Chafi H, Gubichev A, Prat A, Pham MD, Boncz P (2015) The LDBC social network benchmark: interactive workload, In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 619–630
- Faloutsos M, Faloutsos P, Faloutsos C (1999) On power-law relationships of the internet topology. In: ACM SIGCOMM computer communication review, vol 29. ACM, pp 251–262
- Han M, Daudjee K, Ammar K, Özsü MT, Wang X, Jin T (2014) An experimental comparison of pregel-like graph processing systems. *Proc VLDB Endow* 7(12):1047–1058
- Hong S, Depner S, Manhardt T, Lugt JVD, Verstraaten M, Chafi H (2015) Pgxd: a fast distributed graph processing engine. In: Proceedings of international conference for high performance computing, networking, storage and analysis, pp 1–12
- Iosup A, Hegeman T, Ngai WL, Heldens S, Prat-Pérez A, Manhardt T, Chafai H, Capotă M, Sundaram N, Anderson M et al (2016) LDBC graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc VLDB Endow* 9(13): 1317–1328
- Leskovec J, Krevl A (2014) SNAP Datasets: stanford large network dataset collection. <http://snap.stanford.edu/data>
- Leskovec J, Chakrabarti D, Kleinberg J, Faloutsos C (2005a) Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In: Proceedings of the 9th European conference on principles of data mining and knowledge discovery, vol 5, pp 133–145
- Leskovec J, Kleinberg J, Faloutsos C (2005b) Graphs over time: Densification laws, shrinking diameters and possible explanations. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining, pp 177–187
- Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM (2012) Distributed GraphLab: a framework for machine learning in the cloud. *Proc VLDB Endow* 5(8):716–727
- Lu Y, Cheng J, Yan D, Wu H (2014) Large-scale distributed graph computing systems: an experimental evaluation. *Proc VLDB Endow* 8(3):281–292
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of ACM SIGMOD international conference on management of data, pp 135–146
- McSherry F, Isard M, Murray DG (2015) Scalability! but at what cost? In: Proceedings of the 15th USENIX conference on hot topics in operating systems
- Miller GA (1957) Some effects of intermittent silence. *Am J Psychol* 70(2):311–314
- Murphy RC, Wheeler KB, Barrett BW, Ang JA (2010) Introducing the graph 500. Cray Users Group (CUG)
- Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y, Gao W, Jia Z, Shi Y, Zhang S, Zheng C, Lu G, Zhan K, Li X, Qiu B (2014) Bigdatabench: a big data benchmark suite from internet services. In: International symposium on high performance computer architecture, pp 488–499
- Watts DJ, Strogatz SH (1998) Collective dynamics of small-world networks. *Nature* 393(6684): 440–442

Graph Centralities

► Graph Invariants

Graph Compression

► (Web/Social) Graph Compression

Graph Data Integration and Exchange

Angela Bonifati¹ and Ioana Ileana²

¹Lyon 1 University, Villeurbanne, France

²Paris Descartes University, Paris, France

Definitions

Data exchange and data integration are two essential tasks of database interoperability. Data exchange (Fagin et al. 2005) is the problem of translating data structured under a source schema into data adhering to a target schema. Virtual data integration (Lenzerini 2002) on the other hand refers to combining data residing in different sources and providing the user with a unified view of these data, typically by means of a global schema.

Both data exchange and data integration essentially exploit a set of assertions relating elements of the source schema(s) with elements of the target (respectively, global) schema, called *schema mappings*. Such assertions typically are expressed in the form of logical implication (or equivalence) and express query containment (or equivalence) for query expressions over the respective schemas.

An important difference between data exchange and (virtual) data integration is that in data exchange we require the target database to be materialized. Data integration in turn requires no materialization of a global database; the integrated data is simply accessed/computed on the fly, by means of queries.

The fundamental difference between data integration and data exchange in a relational setting and a graph setting is the lack of a precise schema definition. Graph instances typically blend schema information and actual instances with no clear distinction between them. To date, no standard schema language for graphs exists that can be used in a similar fashion as in classical data integration and exchange settings. As such, data integration and exchange represent a substantial departure from their counterparts in

the relational setting, where source and target schemas were clearly separated from source and target instances. Another major difference with respect to the relational setting is the increased expressiveness of the query language used to express the mappings.

Both problems of graph data exchange and data integration have started to be explored in our community. In this entry, we review these previous works. We start by introducing the basic formalism of graph data integration and graph data exchange. We then present the state of the art on the topic and conclude by presenting the open challenges and future directions of research.

G

Overview

In the following, we overview the core elements of graph data exchange and integration, from essential concepts such as graph schemas, instances and queries, up to a brief description of the main problems of interest.

Graph Schemas, Instances, and Queries

A schema for a graph database is a finite alphabet Σ . Let \mathbf{V} be a countably infinite set of node ids. A graph database instance over Σ is a directed, edge-labeled graph $G = (V, E)$, where $V \subset \mathbf{V}$ is a finite set of node ids and $E \subseteq V \times \Sigma \times V$ is a finite set of edges labeled with symbols from Σ . Accordingly, a graph database can also be seen as a relational database comprising binary relations E_a , for $a \in \Sigma$.

Generally, queries over graph databases involve traversal of graph edges while testing for existence of paths respecting certain criteria. Such queries typically use recursion and as such significantly depart from usual relational queries. The classes of queries that have been investigated in the context of graph data exchange and data integration mainly comprise (Florescu et al. 1998; Calvanese et al. 2000b; Wood 2012; Barceló et al. 2012):

- Regular Path Queries (RPQs). These “simplest” binary queries of the form (x, exp, y) retrieve all pairs of nodes linked by a path

whose string satisfies a given regular expression.

- Two-way RPQs (2RPQs). Such queries enhance RPQs with the ability to traverse edges backward.
- Nested Regular Expressions (NREs). These extend 2RPQs with an existential (nesting) operator in the manner of XPath (XPathWorkingGroup 2016), allowing for branching/non-linear navigation.
- Conjunctive RPQs (CRPQs), which are n-ary queries consisting in a conjunction of RPQs.
- Conjunctive 2RPQs (C2RPQs), which are n-ary queries consisting in a conjunction of 2RPQs.
- Conjunctive NREs (CNREs), which are n-ary queries consisting in a conjunction of NREs.

CNREs are an extension of both NREs and C2RPQs, and as such they provide the most general class of queries among the above.

Graph Schema Mappings

Schema mappings are essential ingredients of any data exchange and data integration scenario, since they embody the logical rules relating source and target (respectively, global) schemas. Such rules typically express query containment or equivalence, for queries expressed in the appropriate language; accordingly, for graph databases, rules may feature any class of graph queries, including the six categories listed above.

Graph data exchange mappings. A graph data exchange setting is usually represented by a triple $\Omega = (\Sigma_S, \Sigma_T, \mathcal{M}_{st})$, where Σ_S is the source schema, Σ_T is the target schema, and \mathcal{M}_{st} is the source-to-target mapping. All rules in \mathcal{M}_{st} are of the form (Barceló et al. 2013) $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{x})$, with $\phi_S(\bar{x})$ and $\psi_T(\bar{x})$ being graph queries of matching arity over the source schema Σ_S and, respectively, the target schema Σ_T . Such rules are usually called source-to-target dependencies, and are also (borrowing from the relational setting vocabulary) referred to as source-to-target tuple-generating dependencies (s-t tgds).

A pair of source and target graph databases (G_S, G_T) satisfies a source-to-target dependency $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{x})$ iff $\phi_S(G_S) \sqsubseteq \psi_T(G_T)$. A pair of source and target graph databases (G_S, G_T) satisfies a source-to-target mapping \mathcal{M}_{st} , denoted by $(G_S, G_T) \models \mathcal{M}_{st}$, iff (G_S, G_T) satisfy all the source-to-target dependencies in \mathcal{M}_{st} .

Graph data integration mappings. In a graph data integration setting $\Omega = (\{\Sigma_{S_i}\}, \Sigma_G, \mathcal{M})$ with source schemas Σ_{S_i} and global schema Σ_G , rules in the mapping \mathcal{M} may state query containment of a source query into a global query (*sound sources* (Lenzerini 2002)), in which case one obtains mappings similar to data exchange. However, data integration scenarios may also feature rules stating containment of a global query into a source query (i.e., $\psi(\bar{x}) \rightarrow \phi(\bar{x})$ with ψ over Σ_G and ϕ over Σ_{S_i}), or stating equivalence. Rule and mapping satisfiability semantics are similar to data exchange.

LAV and GAV mappings. Two classes of mappings are of special interest in data integration and data exchange:

- Local As View (LAV) mappings are mappings where in all rules the source query ϕ is atomic, i.e., consists in a single symbol of Σ_S (respectively, Σ_{S_i}).
- Global As View (GAV) mappings are mappings where in all rules the target (respectively, global) query ψ is atomic, i.e., consists in a single symbol of Σ_T (respectively, Σ_G).

To distinguish them from these restricted settings, general (i.e., beyond LAV and/or GAV) mappings have also been called *GLAV mappings*.

Data Exchange Solutions and Universal Representatives

Given a graph data exchange setting $\Omega = (\Sigma_S, \Sigma_T, \mathcal{M}_{st})$ and graph databases G_S over Σ_S and G_T over Σ_T , we say that G_T is a *solution* for G_S under Ω whenever $(G_S, G_T) \models \mathcal{M}_{st}$. The set of solutions of G_S under Ω is denoted $Sol_\Omega(G_S)$.

Due to the semantics of mappings, there may be infinitely many solutions for a given source graph database and data exchange setting. Work on data exchange has therefore placed a focus on the computation of a so-called *universal solution*: a database with incomplete information which intuitively “represents all possible solutions.”

In graph data exchange, Barceló et al. (2013) have introduced a similar concept, namely, the *universal representatives*. These have been defined as graph patterns (Barceló et al. 2011) where the set of nodes N comprises both node ids and labeled nulls, and edges are labeled with expressions without variables over a schema Σ . Such graph patterns model incompleteness of universal representatives in two ways: unknown values for nodes (via the labeled nulls) and imprecise relations between the nodes (as witnessed by the expressions on edges).

A homomorphism from a graph pattern π as above to the graph database $G = (V, E)$ is a mapping $h : N \rightarrow V$ such that (i) h is the identity over $N \cap \mathbf{V}$ and (ii) for each edge of π of the form (u, exp, v) , $(h(u), h(v)) \in Q_{exp}(G)$, where $Q_{exp}(x, y) = (x, exp, y)$. We denote by $Rep_{\Sigma}\pi$ the set of all graph databases G over Σ such that there exists a homomorphism from π to G . Given graph data exchange setting $\Omega = (\Sigma_S, \Sigma_T, \mathcal{M}_{st})$ and graph database G_S over Σ_S , Barceló et al. (2013) then formally define universal representatives as follows: the graph pattern π over Σ_T is a universal representative for G_S under Ω iff $Rep_{\Sigma_T}\pi = Sol_{\Omega}(G_S)$.

Problems of Interest

The main problems under scrutiny for graph data exchange and data integration are the following:

1. *Existence and complexity of computation for data exchange solutions and universal representatives.* In the graph data exchange setting, one is typically interested in both the data complexity (i.e., solely the source database is part of the input) and the combined complexity (i.e., including the schemas and the mapping as input) of such problems. The existence of a solution is of particular interest in the presence of target constraints.

2. *Query answering.* This problem is of interest for both data exchange and data integration. In data exchange, we are interested in answering queries over the target schema, whereas in data integration, queries are stated over the global schema. The semantics adopted for both settings are those of *certain answers*, that is, the intersection of the result sets of the query Q for all data exchange solutions, respectively, all possible global instances compliant with the source instances and the mapping. The query answering problem then consists of deciding, given a query Q over the target (respectively, global) schema, whether a given tuple of node ids belongs to the certain answers of Q . As above, complexity analysis can be carried out in terms of both data complexity (the input consists in the source database(s)) and combined complexity (including the schemas, the mapping, and the query). Oftentimes, the query complexity is also considered.

Query answering is closely related to the universal representatives, since it is often the case that the certain answers for a query can be efficiently obtained by evaluating the query over such representatives. Another important related topic is that of query rewriting, which for data exchange and data integration typically involves reformulating a target (or global) query into a query expressed over the source(s).

Key Research Findings

Query Answering and Query Rewriting

A considerable amount of attention has been dedicated to the problem of query answering and the related problem of query rewriting for graph databases, in particular the large body of work by Calvanese, De Giacomo, Lenzerini, and Vardi (e.g., Calvanese et al. 2000a,b, 2001, 2002, 2003, 2007). An important part of their analysis focuses on view-based query answering and query rewriting. A variety of view-based processing settings for graph databases are placed under scrutiny, including sound/complete/exact views, and complexity results are presented for a variety of query and view languages, including

RPQs, 2RPQs, and C2RPQs, regarding both data and combined complexity. Query answering under sound views in particular, corresponding to a typical LAV setting, is shown to be coNP-complete in data complexity and PSPACE-complete in combined complexity for RPQ queries over RPQ views; these bounds extend to 2RPQs, but the problem reaches EXPSPACE-completeness in combined complexity for the broader class of C2RPQs. A subclass of C2RPQs, called *T2RPQs*, is however identified, where the query body is tree-shaped; for such queries over C2RPQs views, query answering is shown to be PSPACE-complete.

With the work of Calvanese et al. (2012), the authors further go beyond view-based query processing and LAV/GAV mappings, to address sound (i.e., source-to-target only) GLAV mappings, though in a setting restricted to RPQ mappings and RPQ queries. Similar to the LAV results, the paper shows that in such setting query answering is coNP-complete in data complexity and PSPACE-complete in combined complexity, whereas deciding whether a non-empty rewriting exists is EXPSPACE-complete. In addition, the authors show that deciding whether a rewriting of a target query is *perfect*, i.e., it always computes the certain answers when evaluated over the source, is NEXPTIME-complete.

Graph Data Exchange

Barceló et al. (2013) lay the foundations of graph data exchange by mainly focusing on the complexity of query answering and universal representative computation. They also investigate the means of obtaining tractable procedures for solving the above problems of interest.

Barceló et al. (2013)'s study considers the general CNRE class for both queries asked over the target (i.e., those for which certain answers are required) and queries involved in the mappings. The paper is the first to define universal representatives for graph data exchange, corresponding to graph patterns with NRE labeled edges. Furthermore, the authors show how the chase-based procedure used to build universal solutions in the relational setting (Fagin et al. 2005) can

be directly adapted to the graph setting, with data complexity in NLOGSPACE and combined complexity in PSPACE.

As argued by the authors, however, focusing on data complexity is insufficient for graph settings, in particular due to the very large instances typically involved (e.g., social networks, scientific databases). Since the hardness results for combined complexity are a consequence of the hardness of CNRE evaluation, the paper further investigates the restriction of source queries in mappings, from CNREs to the simpler class of NREs. Such *NRE-restricted mappings* comprise only rules of the form $(x, exp, y) \rightarrow \psi_T(x, y)$, with ψ_T a binary CNRE over Σ_T . As shown in the paper, in the presence of NRE-restricted mappings, the universal representative computation becomes tractable: its combined complexity is $O(|G_S|^2 \cdot |\Omega|)$. Similar bounds are shown to hold for acyclic source CNREs.

Concerning query answering, the authors start by emphasizing the difficulty of the certain answers problem. Indeed, while the problem has been shown to be PSPACE-complete in combined complexity for RPQ queries and mappings, it is shown to be EXPSPACE-complete for CNRE queries and mappings. On the other hand, going from RPQs to CNREs keeps data complexity in coNP, but the authors refine the lower bound by exhibiting a setting featuring RPQ mappings and a “word” RPQ queries, for which the problem is shown to be coNP-complete. Since word RPQ query can be translated into relational conjunctive queries, such setting shows how query answering in graph data exchange importantly diverges from the relational setting, where computing certain answers to conjunctive queries is known to be tractable in data complexity (Fagin et al. 2005). The hardness results shown also point toward the insufficiency of NRE-restricted mappings for achieving feasible query answering.

The pursuit of tractability then leads the authors to examining further restrictions on the mappings and the queries. A new restriction on the target queries in the mappings leads to the definition of *rigid mappings*, where the right-

hand side does not employ disjunction or recursion. Such mappings in turn are useful for achieving tractability in several cases. For an NRE query exp and mappings that are both NRE-restricted and rigid, the combined complexity of query answering is indeed shown to be $O(|G_S|^2 \cdot |\Omega| \cdot |exp|)$, by relying on the universal representative computation. If one further restricts the setting to mappings that are NRE-restricted and GAV, the combined complexity for NRE query answering becomes linear, i.e., $O(|G_S| \cdot |\Omega| \cdot |exp|)$, this time by employing query rewriting techniques. Moreover, if one only examines data complexity, in the case of CNREs and rigid mappings, query answering can be solved in polynomial time.

Last but not least, the authors initiate the investigation of tractability in data complexity for answering *tame CRPQs*, which are CRPQs that do not employ any Kleene star, over mappings that do not have existential quantifiers on the right-hand side (e.g., NRE mappings), thus leading to universal representatives without labeled nulls. For such queries and mappings, the authors derive two interesting conditions under which query answering becomes tractable and further prove the maximality of the shown classes w.r.t. to tractability.

Target Constraints

A typical relational data exchange setting (Fagin et al. 2005) comprises, in addition to the source and target schemas and the source-to-target mapping, a set of rules applicable to the target database, called *target constraints*. Contrarily to the relational setting, in which target constraints were essential components of a scenario, these constraints have not been included in the foundational definitions of (Barceló et al. 2013), and have been so far poorly adopted in graph data exchange.

Borrowing from the relational setting (Beeri and Vardi 1984, Fagin et al. 2005), one may consider graph data exchange scenarios comprising two classes of target constraints, namely:

- *target tuple-generating dependencies* (or simply *t tgds*), i.e., $\phi_T(\bar{x}) \rightarrow \psi_T(\bar{x})$, with ϕ_T and ψ_T being graph queries of matching arity over the target schema Σ_T
- *target equality-generating dependencies* (or simply *t egds*), i.e., $\phi_T(\bar{x}) \rightarrow (x_1 = x_2)$, with ϕ_T being a query over the target schema Σ_T and x_1, x_2 among the variables of \bar{x} .

Target tgds have the same query containment semantics as s-t tgds, whereas target egds express equalities among node ids, and as such intuitively involve *merging* nodes in the data exchange solution. As is the case in the relational setting, target constraints have an important influence upon the existence of a data exchange solution.

Boneva et al. (2015) have added limited classes of target constraints to the setting of Barceló et al. (2013) by focusing on t egds and limited forms of t tgds. The paper studies the complexity of existence of solutions and query answering with the assumption that the source schema and instance are fixed (query complexity). Another hypothesis made in their work is to be able to export relational tuples from the source rather than graph patterns. The results are quite negative in the sense that both existence of solutions and query answering in the presence of arbitrary t egds and t tgds are shown to be NP-hard and coNP-hard, respectively. Because of this intractability and the difficulty of equating two constants in case of t egds, the authors then focus on a simpler class of constraints. These are called same-as tgds, and they correspond to full GAV t tgds in which the right-hand side contains a unique binary atom *sameAs*. Adding these constraints makes the existence of solutions trivial as it suffices to compute the solutions with s-t tgds as in Barceló et al. (2013) and then add the corresponding tuples to the *sameAs* relation. However, even with this limited fragment of target constraints, query answering stays coNP-hard.

Mapping Management for Data Graphs

The recent work of Francis and Libkin (2017) enriches the landscape of graph mapping management studies by considering *data graphs*. These

are edge-labeled graphs where furthermore vertices are adorned with *data values*. Formally, letting \mathbf{V} be a countably infinite set of node ids, and \mathbf{D} a countably infinite set of values, a node in a data graph is a pair $(v, d) \in \mathbf{V} \times \mathbf{D}$. A data graph is then of the form $G = (V, E)$ with $V \subseteq \mathbf{V} \times \mathbf{D}$ and $E \subseteq V \times \Sigma \times V$.

Data graphs are very suitable for capturing property graphs, which are the models employed by some of the existing graph database management systems, such as Neo4j (NeoTechnology 2017). As shown by the authors, however, adding data to the picture makes query answering sensibly harder than in the standard (i.e., without data adornment) graph database case.

Francis and Libkin (2017) consider mappings based on RPQs and in turn investigate query answering for *data RPQs*, which were defined in Libkin et al. (2015, 2016). These are extensions of RPQs with regular expressions that mix alphabet letters (i.e., edge labels) and data values, in other words queries that mix navigational properties and data. The three main classes of data RPQs under scrutiny are (i) data path queries, (ii) regular expressions with equality (REEs, or equality RPQs), and (iii) regular expressions with memory (REMs, or memory RPQs).

A first strong result of the paper shows that query answering is in general undecidable in data complexity for data RPQs (with a notable exception for the data path queries and REEs without inequalities, for which it is in coNP). The undecidability is moreover shown to hold even in a simple setting comprising a relational/reachability LAV mapping and an equality RPQ.

As identified by the authors, the issue comes precisely from the reachability target queries and thus suggests that in order to recover decidability one needs to focus on mappings where the target queries exclude any form of recursive navigation and only consist in word RPQs. For such mappings, called “relational mappings”, query answering becomes coNP in data complexity and PSPACE in combined complexity. Moreover, the problem becomes NLOGSPACE in data complexity for relational mappings and data path queries comprising at most one subexpression of

the form $c \neq$ retrieving paths whose extremities have different data values.

Pursuing the search for tractability while restricting their focus to relational mappings, the authors investigate two very interesting alternatives for efficient query answering for data RPQs by relying on universal- and canonical-like solutions. For restrictions of REMs and REEs to equalities (no inequalities allowed), the authors show that query answering can be solved by relying on the construction of a so-called *least informative solutions*, which employs labeled nulls for both node ids and data values, in a manner similar to standard relational data exchange canonical universal solutions. Accordingly, data complexity for query answering becomes NLOGSPACE for all the considered queries, whereas combined complexity becomes PSPACE for restricted REMs and PTIME for restricted REEs. A very interesting take further concerns data RPQs in general, without restrictions. It involves providing an *under-approximation* of a query’s result set by relying on universal solutions that employ SQL-like nulls (null value for which all equalities and inequalities evaluate to *false*) for the data on nodes. Query answering over such a null-based universal solution is shown to be NLOGSPACE in data complexity and PSPACE, respectively PTIME in combined complexity for REMs, respectively REEs.

Future Directions for Research

Despite the early work on graph data integration and exchange surveyed in this entry, several open problems can be identified and might deserve the attention of our community in the next future. In particular, for what concerns the coverage of the characteristics of the data exchange setting, one missing component is certainly embodied by target constraints, which have been poorly exploited so far. The definition of universal representatives as in Barceló et al. (2013) cannot be easily adopted in the case of target constraints (Boneva et al. 2015). Indeed, even for full GAV

target tgds, query answering stays intractable (Boneva et al. 2015). Suitable variants of universal representatives, possibly coupled with target constraints, need to be conceived. Proper subclasses of target egds, such as graph keys (Fan et al. 2015), graph functional dependencies, and graph entity dependencies (Fan and Lu 2017), might be relevant for envisioning practical algorithms for data exchange in the presence of these target constraints and in the spirit of Bonifati et al. (2016) for the relational setting. Another important missing component consists of graph schemas, which are presently exemplified as alphabets of graph labels as discussed in this entry. More sophisticated schemas for RDF based on ShEx expressions (ShExCommunityGroup 2017) have been designed, and their adoption in graph data integration and exchange is worth exploring. For what concerns query rewriting, more expressive classes of graph constraints beyond RPQs, 2RPQs, C2RPQs are worth exploring. Due to the high complexity of query answering and rewriting even for basic fragments such as RPQs, alternative Datalog-based rewritings beyond certain answers and computable in polynomial time (Francis et al. 2015) are also desirable for more expressive fragments of graph queries. Finally, the basic fragment of graph queries considered in schema mappings consists of CNRE queries, which correspond to their relational counterparts (CQs). However, in a more comprehensive graph data integration scenario, one can adopt more expressive languages, such as Regular Queries (RQs) (Reutter et al. 2017), which is the largest fragment of graph queries for which containment has been shown to be decidable. The definition of more expressive graph data models, such as the Property Graph data model (Angles et al. 2017), and suitable query languages thereof will also bring fresh valuable insights to graph data integration and exchange along the line of research initiated in Francis and Libkin (2017). Investigating the problems of interest discussed in this entry for queries expressed in modern graph query languages is also an interesting direction for future work.

Cross-References

- ▶ [Graph Path Navigation](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)
- ▶ [Graph Data Models](#)

References

- Angles R, Arenas M, Barceló P, Hogan A, Reutter JL, Vrgoc D (2017) Foundations of modern query languages for graph databases. *ACM Comput Surv* 50(5):68:1–68:40
- Barceló P, Libkin L, Reutter JL (2011) Querying graph patterns. In: Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2011, Athens, 12–16 June 2011, pp 199–210
- Barceló P, Pérez J, Reutter JL (2012) Relative expressiveness of nested regular expressions. In: Proceedings of the 6th Alberto Mendelzon international workshop on foundations of data management, Ouro Preto, 27–30 June 2012, pp 180–195
- Barceló P, Pérez J, Reutter JL (2013) Schema mappings and data exchange for graph databases. In: Joint 2013 EDBT/ICDT conferences, ICDT'13 proceedings, Genoa, 18–22 Mar 2013, pp 189–200
- Beeri C, Vardi MY (1984) A proof procedure for data dependencies. *J ACM* 31(4):718–741
- Boneva I, Bonifati A, Ciucanu R (2015) Graph data exchange with target constraints. In: Proceedings of the workshops of the EDBT/ICDT 2015 joint conference (EDBT/ICDT), Brussels, 27 Mar 2015, pp 171–176
- Bonifati A, Ileana I, Linardi M (2016) Functional dependencies unleashed for scalable data exchange. In: Proceedings of the 28th international conference on scientific and statistical database management, SSDBM 2016, Budapest, 18–20 July 2016, pp 2:1–2:12
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2000a) View-based query processing and constraint satisfaction. In: 15th annual IEEE symposium on logic in computer science, Santa Barbara, 26–29 June 2000, pp 361–371
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2000b) View-based query processing for regular path queries with inverse. In: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, Dallas, 15–17 May 2000, pp 58–66
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2001) View-based query answering and query containment over semistructured data. In: Database programming languages, 8th international workshop, DBPL

- 2001, Frascati, 8–10 Sept 2001, Revised Papers, pp 40–61
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2002) Rewriting of regular expressions and regular path queries. *J Comput Syst Sci* 64(3):443–465
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2003) Reasoning on regular path queries. *SIGMOD Rec* 32(4):83–92
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2007) View-based query processing: on the relationship between rewriting, answering and losslessness. *Theor Comput Sci* 371(3):169–182
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2012) Query processing under GLAV mappings for relational and graph databases. *PVLDB* 6(2): 61–72
- Fagin R, Kolaitis PG, Miller RJ, Popa L (2005) Data exchange: semantics and query answering. *Theor Comput Sci* 336(1):89–124
- Fan W, Lu P (2017) Dependencies for graphs. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems, PODS 2017, Chicago, 14–19 May 2017, pp 403–416
- Fan W, Fan Z, Tian C, Dong XL (2015) Keys for graphs. *PVLDB* 8(12):1590–1601
- Florescu D, Levy AY, Suciu D (1998) Query containment for conjunctive queries with regular expressions. In: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Seattle, 1–3 June 1998, pp 139–148
- Francis N, Libkin L (2017) Schema mappings for data graphs. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems, PODS 2017, Chicago, 14–19 May 2017, pp 389–401
- Francis N, Segoufin L, Sirangelo C (2015) Datalog rewritings of regular path queries using views. *Log Methods Comput Sci* 11(4), pp 1–27
- Lenzerini M (2002) Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Madison, 3–5 June 2002, pp 233–246
- Libkin L, Tan T, Vrgoc D (2015) Regular expressions for data words. *J Comput Syst Sci* 81(7):1278–1297
- Libkin L, Martens W, Vrgoc D (2016) Querying graphs with data. *J ACM* 63(2):14:1–14:53
- NeoTechnology (2017) The Neo4J open source edition. <https://github.com/>, <https://github.com/neo4j/neo4j/releases/tag/3.3.0>
- Reutter JL, Romero M, Vardi MY (2017) Regular queries on graph databases. *Theory Comput Syst* 61(1):31–83
- ShExCommunityGroup (2017) ShEx: Shape Expressions. <https://www.w3c.org>, <http://shex.io/>
- Wood PT (2012) Query languages for graph databases. *SIGMOD Rec* 41(1):50–60
- XPathWorkingGroup (2016) XML Path Language (XPath) 2.0 Second Edition. <https://www.w3c.org>, <https://www.w3.org/TR/xpath20/>

Graph Data Management Systems

Marcus Paradies^{1,2}, Stefan Plantikow³, and Oskar van Rest⁴

¹SAP SE, Walldorf, Germany

²SAP SE, Berlin, Germany

³Neo4j, Berlin, Germany

⁴Oracle, CA, USA

Classification

The abundance and diversity of massive-scale, graph-structured data and the ever-growing interest of large enterprise companies to analyze them are the key drivers of the recent advances in graph data management research. Further, graph data management has witnessed a steady increase in applications from various industry domains, such as social media, logistics and transportation, gas and oil utility networks, finance, public security, and the pharmaceutical industry.

The widespread adoption of graph technology and the diversity in requirements for the aforementioned application domains are the key drivers for the development of a variety of graph data management systems (GDMSS). These GDMSS are typically tailored to a specific graph workload, which is distinctive to the application domain and drives the design of the core system components, such as the *primary/secondary storage*, the *query execution engine*, and the *user interface(s)*.

From a data model perspective, currently two graph data models (Angles and Gutierrez 2008) receive the widest adoption in GDMSS: the *property graph* data model (PGM) and the RDF data model. While the PGM data model has recently been adopted by various GDMSS, there is also still a growing interest in specialized GDMSS and frameworks supporting the RDF data model and the accompanying SPARQL query language (W3C 2013; Stardog 2017; Erling 2012; Ontotext 2017; Neumann and Weikum 2008; AllegroGraph 2017; MarkLogic 2017; Jena 2017; RDF4J 2017). Few GDMSS offer support

for both, RDF and PGM, by exposing data model-specific query languages and transformations between them (Oracle 2017a; Amazon Neptune 2017).

GDMSS can be broadly classified into *native* and *nonnative* systems. Often GDBMSS provide different degrees of *nativeness* by choosing carefully which core system components should be graph-aware. A GDMS is considered to be *native*, if it (1) provides fast data access—often referred to as *index-free adjacency access*—(2) and a graph-specific query runtime, including graph-specific query optimization and processing strategies, (3) and if it exposes a graph data model, such as the RDF or the property graph data model, and corresponding access methods or query languages directly to the user.

Fully native GDBMSS implement graph-aware components for all three core components (Neo4j 2017; TigerGraph 2017; MemGraph 2017; Sevnenich et al. 2016; Amazon Neptune 2017; Tanase et al. 2014; Cayley 2017; Datastax 2017). In contrast, partially native GDBMSS provide graph-aware implementations for selected components, but typically lack a deep integration of graph-specific components within the core database engine. Most RDBMS are only to some extent *graph-aware*, most commonly by exposing the graph data model by some user interface to the end user (Oracle 2017a; Rudolf et al. 2013; Simmen et al. 2014; Sun et al. 2015; Microsoft 2017; AgensGraph 2017). Moreover, partially native GDBMSS include *multi-model* databases, which combine different data models within a single system (OrientDB 2017; ArangoDB 2017).

Each of the GDBMSS is tailored to specific graph workloads, which are the key drivers for the system design and the level of *graph awareness*. In the following, we discuss two major graph workloads that are predominant in realistic graph use cases.

Operational Graph Processing The ability to handle graph topologies where the structural shape of vertices and edges evolves over time is a fundamental requirement of an operational GDMS. *Structural changes* describe modifications to the structure of a single vertex/edge, i.e., the

addition, manipulation, and removal of attributes on vertices and edges. Especially use cases that target social networks and communication networks demand a flexible and scalable graph persistence that can cope with frequent updates to vertex/edge attributes as well as changes to the graph topology. By *topological changes* we refer to modifications of the graph topology, i.e., the addition and removal of vertices and edges. In addition to the ability to modify the graph, an operational GDBMS also must provide transactional guarantees comprising atomicity, consistency, isolation, and durability. A query workload in such an operational scenario often performs short-running queries accessing only a small portion of the graph. This often includes graph pattern matching queries, which aim at finding query embeddings in the data graph.

Graph Analytics Graph analytics workloads often consist of tasks that are computationally intensive and typically take seconds, minutes, or sometimes hours to complete on large graphs. This is in contrast to operational graph processing where queries are typically short-running and often only consume little time to complete. To target these computationally intensive and mostly read-only analytical workloads, specialized systems have been created that provide efficient data structures for storing relatively static graphs and, typically, process graph algorithms or graph queries in a parallel and/or in a distributed fashion. Because such systems do not always provide strong transactional guarantees, they are typically called “graph analytics toolkits” rather than graph databases.

Storage Representations

Depending on their level of *graph awareness*, GDBMSS implement different storage representations. Broadly speaking, the design space can be divided into *native* and *nonnative* graph representations.

Native, Read-Only/Read-Mostly Immutable, native graph representations tailored for

read-mostly workloads are often implemented using a CSR data structure (Saad 2003; Brameller et al. 1976). A CSR data structure has the advantage that it provides large temporal and space locality when accessing the neighborhood of a vertex. This is often observed for in-memory GDBMSS, such as ORACLE PGX (Sevenich et al. 2016; Hong et al. 2015) and SYSTEM-G (Tanase et al. 2014). If the graph is too large to fit into the main memory of a single machine, suitable alternatives are processing the graph on cheaper yet fast SSDs, applying compression techniques on the graph data or by sharding the graph data across multiple machines in a cluster (Tanase et al. 2014).

Updates to the CSR data structure can be either applied in batch updates (Haubenschmid et al. 2016) or by snapshotting the CSR data structure (Macko et al. 2015).

Native, Transactional NEO4J is the original, prevalent system for native, transactional processing of property graphs. Its graph representation is based on the notion of index-free adjacency, which provides $\mathcal{O}(1)$ cost for traversing a relationship and has been implemented using a proprietary set of record files. It uses direct offset pointers for navigational data access to avoid the additional lookup costs of tree data structures. The record files utilize additional pointers to reduce the overhead of traversing dense nodes as well as generally minimize the amount of records that need to be accessed during the graph traversal. The main storage is complemented by additional indexes for direct by-value lookup and full-text search.

Relational, Read-Only The OPENCYPHER project recently released CYPHER FOR APACHE SPARK (openCypher implementers group 2017a), a new implementation of OPENCYPHER that optimizes and compiles queries for execution by the relational Catalyst engine of APACHE SPARK SQL. Graphs are represented as sets of disjoint scan tables for nodes and relationships with a given label. Using the immutable and lazily materialized tables of APACHE SPARK naturally provides snapshotting, partial caching, and partitioning capabilities. This graph representation relies

on the use of a graph schema since the scan tables themselves are expected to be statically typed by APACHE SPARK.

Relational, Transactional A graph can be seamlessly represented as a set of vertex/edge tables, where each record in a vertex table corresponds to a single vertex and each record in an edge table represents a single edge (Rudolf et al. 2013; Simmen et al. 2014; Oracle 2017a). Each vertex and edge attribute is mapped to a single column in the corresponding vertex and edge tables, respectively. For graphs without or with only few vertex or edge attributes, a database consisting of two universal tables—one for vertices and one for edges—is a commonly used physical representation.

Recently, non-relational extensions of RDBMSS have been applied to encode graph topology information (Microsoft 2017) and vertex/edge properties using JSON document storage functionality (Sun et al. 2015).

Similarly, RDF GDBMSS often rely on relational storage layouts to represent RDF triples through universal tables (Erling 2012) and by using exhaustive, relational indexing techniques (Neumann and Weikum 2008).

Query Languages

Graphs are queried using both declarative and imperative languages. Declarative languages are used by operational DBMSS due to their ease of use and the ability to rely on a vast body of research for query optimization, while imperative languages are used for graph analytics workloads due to their versatility and the provided high level of manual control over query execution.

Declarative QLs

Graph pattern matching, i.e., finding and filtering instances of connected nodes and relationships, is the central functionality of declarative graph query languages. Pattern matching may be expressed using visual “Ascii-Art” graph patterns as pioneered by OPENCYPHER or via explicit recursion. Typically the connections in a pattern are described in terms of a regular expression

over the labels along a matched path. This is called a regular path query (RPQ).

Languages use different pattern matching semantics: Homomorphic matching returns all matches but may lead to infinite results. Isomorphic matching prevents this by disallowing matching nodes or relationships per match but is computationally intractable for certain patterns. Single shortest path matching is tractable and never leads to infinite results, but is non-deterministic.

Languages additionally provide fundamental relational capabilities, such as aggregation, slicing, or sorting of results.

OPENCYpher 9 (Francis et al. 2017; openCypher implementers group 2017b) is the language of the **OPENCYpher** community. Originally developed by NEO4J as a more high-level alternative to **GREMLIN**, it is implemented by multiple vendors (SAP HANA, REDIS, AGENS GRAPH for Postgres, CYPHER FOR APACHE SPARK), as well as research projects. **OPENCYpher** provides graph pattern matching using isomorphism, relational capabilities (including tabular query composition, basic subqueries, query parameters, and calling external algorithms), single and all shortest path matching and all path enumeration using first-class path values, as well as both an update and a schema definition language. The current **OPENCYpher 9** supports no alternation between non-atomic labels. However full RPQs and path patterns have been specified as a separate language extension. The next version of the language, **OPENCYpher 10**, is currently developed with support for a richer type system, more powerful subqueries, and graph query composition for passing graphs between queries and views.

PGQL (Property Graph Query Language) (van Rest et al. 2016; Oracle 2017b) is a pattern matching query language developed at **ORACLE** and is closely aligned to **SQL**. Its latest version (Oracle 2017b) has powerful regular path expressions with arbitrary filter predicates on vertices and edges along paths. Its pattern matching semantic is based on homomorphism, and the language provides basic relational support (ORDER BY, SELECT, etc.).

SQL SERVER provides an extension to **SQL** that allows for creating a graph by turning existing relational tables into vertex or edge tables and provides graph querying support through specification of fixed size graph patterns (Microsoft 2017).

SPARQL (W3C 2013) is the declarative query language used for the **RDF** data model. It provides pattern matching, basic relational capabilities, as well as constructing and returning graphs.

G-CORE (Angles et al. 2017) is a language designed as a community effort between industry and academia, meant to guide the evolution of existing and future graph query languages. **G-CORE** is composable, meaning that graphs are the input and the output of queries, and treats paths as first-class citizens.

Traversals form the foundation of **GREMLIN** (Rodriguez 2015), a language that organizes graph computation as a series of steps for pattern matching, sorting, aggregating, and data flow logic, as well as a step for basic pattern matching and even graph computation. **GREMLIN** was originally closely tied to the Groovy programming language. Recent versions compile to a bytecode that is shared between implementations. However real-world **GREMLIN** often also contains inline logic written in the client programming language.

Imperative QLs

Imperative languages have been the dominant paradigm for analytical graph processing in high-performance computing but also in handwritten applications (e.g., Boost Graph Library for C++). Additionally, many classical graph algorithms were originally developed in an imperative setting, and therefore, many systems have chosen an imperative query language to enable applications to implement such algorithms.

Low Level

Low-level, imperative query languages expose an imperative API to directly access the graph in a fine-granular manner.

Vertex-centric systems such as **PREGEL** (Malewicz et al. 2010) and **SIGNAL/COLLECT** (Stutz et al. 2010) rely on a vertex-centric message-passing abstraction to perform

massively parallel asynchronous or synchronous graph computations. Each vertex executes a user-provided `compute()` function (the query), which receives messages from neighbors, updates internal state, and sends messages to neighbors as well as result collectors to advance the computation until the result converges.

Imperative APIs like BLUEPRINTS (originated at NEO4J) enable programs written in an imperative, general-purpose programming language to directly access the graph. A key development of such APIs is the notion of a traversal: a semi-declarative description of how to search the graph starting from a fixed vertex using different search strategies (BFS, DFS) and user-provided filtering predicates.

High Level

GRAPHSCRIPT (Paradies et al. 2017) is a domain-specific, read-only graph query language tailored to serve advanced graph analysis tasks and to ease the specification of custom, complex graph algorithms. GRAPHSCRIPT is statement-based and provides high-level, graph-specific language constructs to formulate graph algorithms in a user-friendly and intuitive manner while offering competitive execution performance compared to manually tuned graph algorithms in a low-level programming language. It facilitates an imperative programming model with control flow elements and graph querying operations.

GREEN-MARL (Hong et al. 2012, 2013), currently being developed at ORACLE LABS, is among the first domain-specific graph query languages targeting graph analytics. The language has *graph*, *vertex*, and *edge* types as first-class citizens and exposes high-level constructs for graph traversal and iteration, such as breadth-first search (BFS), depth-first search (DFS), and incoming and outgoing neighbor iteration.

Graph Processing

On a high level, current graph processing approaches either fall into the category of “native graph processing,” in which systems are built

from scratch and optimized specifically for graph processing, or “relational graph processing,” in which existing RDBMSs or tools are repurposed and extended for graph processing.

Native Graph Processing

Besides the storage aspect, as discussed before, there are three other important aspects to consider when designing a native GDMS.

Compilation Strategies For processing of graph *queries* (e.g., SPARQL, OPENCYpher, or PGQL queries), the typical approach of GDMSS is that of cost-based query optimization, which is similar to the approach in traditional RDBMSs. Dynamic programming is used to explore the plan space in order to estimate a cost for the various possible plans using a set of heuristics, precomputed data statistics, or through dynamic data sampling. However, graph queries impose additional challenges since they are often recursive in nature and estimation errors increase even faster than for relational queries that typically only have a fixed number of joins. There are systems (e.g., APACHE SPARK) that are adopting mid-query re-optimization to overcome such problems.

For processing of graph *algorithms* (e.g., GRAPHSCRIPT or GREEN-MARL procedures), the typical approach is that of code generation rather than interpretation because the overhead of compiling generated code is typically neglectable as graph analytics algorithms are often computationally intensive and have long-living executions. In ORACLE PGX, GREEN-MARL procedures are translated into optimized and parallel JAVA or C++ programs through code generation. In SAP HANA, GRAPHSCRIPT procedures are directly compiled into parallelized, low-level, LLVM-based executable code and cached for later usage. GRAPHSCRIPT generates code, which directly accesses internal data structures and APIs, thereby eliminating function calls and property-type interpretations. NEO4J provides the OPENCYpher language for querying data as well as a programmatic API for imperative graph processing. OPENCYpher queries are translated into an operator tree that is

a mix of relational and graph-specific operators for graph traversal and that is executed using both interpretation and compilation to Java bytecode.

In-Memory Processing To avoid disk I/O being the bottleneck in graph processing, graphs are often fully or partially loaded into memory. This is especially meaningful for the structural data of the graph, as edge traversal is a crucial operation in graph processing. Nonstructural data, such as labels and properties on vertices or edges, may be partially loaded into memory, depending on available space. In ORACLE PGX, graphs are loaded into memory in their entire, and scaling out is necessary (i.e., partition the graph over multiple machines) when graphs are too large to fit into the memory of a single machine. SAP HANA allows fully or partially loading vertex and edge tables on a column level into memory. Lightweight compression techniques aim at keeping the memory footprint low. For GRAPHSCRIPT, static code analysis is used to determine the accessed vertex/edge properties and only load the corresponding subgraph into memory. In NEO4J, graphs are primarily stored on disk and mapped into memory using a custom page cache, which is optimized for concurrent, low latency, fast random access graph traversal. In practice, the majority of the graph working set of a database cluster machine usually resides in memory.

Parallel Processing Parallel processing of graphs is necessary to make full advantage of available multi-threading capabilities of modern computing architectures. In transactional graph workloads, there are often large amounts of short-living tasks, so multiple tasks are executed concurrently. However, in analytical graph workloads, there are typically fewer but heavier longer-living tasks, so the typical strategy is that of *intra-task* parallelization rather than *inter-task* parallelization. In Neo4j, OLTP graph data is loaded into separate, specialized in-memory data structures for efficient parallel graph analytical processing. Traversals in graph queries or algorithms can naturally be processed in a parallel breadth-first search manner, such that at each step, the available threads are

given an equal number of partial solutions and threads synchronize at the end of each step to produce a new set of partial solutions for the next step (Raman et al. 2014; Hong et al. 2012). Although easily parallelizable, the breadth-first search approach may suffer from a large memory footprint, which can become an issue in graph query processing, as the number of partial solutions can grow exponentially to the size of the graph. Therefore, depth-first search processing, which can be parallelized through work-stealing techniques, may provide a suitable alternative (Roth et al. 2017).

Indexing Techniques Graphs often have labels on vertices and edges for indicating types (e.g., “Person” or “knows”) and also have properties on vertices or edges for providing unique identifiers (e.g., an “email” or “ISBN” property). Because these labels and identifiers are frequently used in filter predicates of graph queries or graph algorithms, they require specialized indexes to speed up processing. For example, in ORACLE PGX and NEO4J, secondary indexes can be created, which retrieve vertices with a particular label in constant time. Furthermore, there is an index for mapping a vertex identifier to the respective vertex, again, in constant time. Both indexes can help avoid scanning over the entire set of vertices. NEO4J always maintains indexes for retrieving vertices with certain labels. Furthermore, NEO4J allows the creation of indexes on properties of nodes with a given label, as well as additional index structures for full-text indexing and globally indexing all properties.

In addition to the primary data location as relational tables, transient secondary index structures, such as adjacency lists or CSR, can be used to accelerate compute-intensive graph analytics tasks by creating them prior to the query execution (Hauck et al. 2015). Since the majority of complex graph algorithms and workflows perform heavyweight, long-running computations on the entire graph, the overhead for the index construction is outweighed by the actual execution time of the task.

Besides indexes that are created ahead of query execution, there are also indexes that

are created during query execution. Some indexing techniques are based on “supernodes” (or “landmarks”), which are vertices with a disproportionate number of incoming or outgoing edges, and are common in real-world graphs. Since the supernodes are well connected, they are traversed many times and computations are cached to avoid expensive recomputations (Valstar et al. 2017).

Relational Graph Processing

Naturally, existing RDBMSS are being extended to process graph data. SAP HANA is the first full-fledged RDBMS that tightly integrates graph processing into the database kernel, allowing to seamlessly combine relational and graph operations within the same database engine (Rudolf et al. 2013; Paradies et al. 2015). OPENCYPHER queries are translated into relational operators, while for processing of GRAPHSCRIPT, there is an additional set of graph-specific operators, such as graph traversal, shortest path computation, and subgraph induction, that are not mapped to relational operators but instead implemented as native graph operators in the RDBMS execution engine (Paradies et al. 2017).

SQLGRAPH (Sun et al. 2015) translates GREMLIN into a combination of pure SQL functions, user-defined functions, common table expressions, and stored procedures. The latter ones are only used for graph updates without side effects and for recursive traversal queries without an explicit recursion depth boundary.

TERADATA ASTER (Simmen et al. 2014), a large-scale analytics platform, provides an iterative vertex-oriented JAVA-based programming abstraction, which can be used to implement bulk-synchronous graph algorithms. Graph analytics functions, both pre-built and user-implemented ones, are modeled as polymorphic table operators that can be invoked directly from SQL.

VIRTUOSO (Erling 2012) is a hybrid columnar relational/graph DBMS with a SPARQL interface. Internally, VIRTUOSO translates SPARQL queries into SQL with graph-specific extensions, such as the computation of the transitive closure using a dedicated operator.

CYPHER FOR APACHE SPARK (openCypher implementers group 2017a) is a new implementation of Cypher atop the relational Catalyst engine of APACHE SPARK. CYPHER FOR APACHE SPARK is noteworthy in providing experimental support for graph query composition and working with multiple graphs via proposed OPENCYPHER 10 language extensions (openCypher implementers group 2017b). Graphs are directly transformed into new graphs by transforming the underlying immutable scan tables.

Discussion

GDMSS are very new, and various advances in storage representation, query interfaces, and processing techniques are to be expected. In particular, we see four trends that will significantly influence the design and implementation choices of upcoming single-node graph processing systems. (1) We observe an increasing interest in the uses of *high-level graph query languages* for graph analysis. (2) We see a convergence toward *hybrid transactional/analytic graph processing*, which combines graph analysis as performed by highly tuned graph processing systems with transactional graph processing as provided by GDMSS or traditional RDBMSS. (3) We observe that existing systems are moving toward becoming *multi-model systems* (relational, RDF, PGM), as evidenced by GDMSS adopting relational features and RDBMSS being extended with graph capabilities. This is supported by convergence on a *common definition of the property graph model* between different languages that is closely related to the ongoing discussions on working with multiple graphs, graph query composition, and schema models for graph data. (4) We observe an interesting line of research on *compressed and skew adaptive storage* of graphs, which is important since efficiency of memory bandwidth usage is an ongoing challenge in large multiprocessor machines.

References

- AgensGraph (2017) <http://bitnine.net/>
- AllegroGraph (2017) <https://franz.com/agraph/allegrograph/>
- Amazon Neptune (2017) <https://aws.amazon.com/neptune/>
- Angles R, Gutierrez C (2008) Survey of graph database models. *ACM Comput Surv (CSUR)* 40(1):1–39
- Angles R, Arenas M, Barcelo P, Boncz P, Fletcher GHL, Gutierrez C, Lindaaker T, Paradies M, Plantikow S, Sequeda J, van Rest O, Voigt H (2017) G-CORE: a core for future graph query languages. <https://arxiv.org/abs/1712.01550>
- ArangoDB (2017) <https://www.arangodb.com/>
- Brameller A, Allan RN, Hamam Y (1976) Sparsity: its practical application to systems analysis. Pitman, London
- Cayley (2017) <https://cayley.io/>
- Datastax (2017) <https://www.datastax.com/products/datastax-enterprise-graph>
- Erling O (2012) Virtuoso, a hybrid RDBMS/graph column store. *IEEE Data Eng Bull* 35(1):3–8
- Francis N, Green A, Guagliardo P, Libkin L, Lindaaker T, Marsault V, Plantikow S, Rydberg M, Selmer P, Taylor A (2017, Submitted) Cypher: an evolving query language for property graphs
- openCypher implementers group T (2017a) Cypher for Apache Spark. <https://github.com/opencypher/cypher-for-apache-spark>
- openCypher implementers group T (2017b) Cypher query language reference, version 9. <https://github.com/opencypher/openCypher/blob/master/docs/openCypher9.pdf>
- Haubenschmid M, Then M, Hong S, Chafi H (2016) ASGraph: a mutable multi-versioned graph container with high analytical performance. In: Proceedings of the fourth international workshop on graph data management experiences and systems, GRADES’16, pp 8:1–8:6
- Hauck M, Paradies M, Fröning H, Lehner W, Rauhe H (2015) Highspeed graph processing exploiting main-memory column stores. In: Proceedings of the Euro-Par 2015 international workshops, pp 503–514
- Hong S, Chafi H, Sedlar E, Olukotun K (2012) Green-Marl: a DSL for easy and efficient graph analysis. In: Proceedings of the 17th international conference on architectural support for programming languages and operating systems, ASPLOS’12, pp 349–362
- Hong S, Rodia NC, Olukotun K (2013) On fast parallel detection of strongly connected components (SCC) in small-world graphs. In: International conference for high performance computing, networking, storage and analysis, SC’13, pp 92:1–92:11
- Hong S, Depner S, Manhardt T, Van Der Lugt J, Verstraaten M, Chafi H (2015) PGX.D: a fast distributed graph processing engine. In: Proceedings of the international conference for high performance comput-
- ing, networking, storage and analysis, SC’15, pp 58:1–58:12
- Jena A (2017) <https://jena.apache.org/>
- Macko P, Marathe VJ, Margo DW, Seltzer MI (2015) LLAMA: efficient graph analytics using large multi-versioned arrays. In: Proceedings of the 31st IEEE international conference on data engineering, ICDE’15, pp 363–374
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the ACM SIGMOD international conference on management of data, pp 135–146
- MarkLogic (2017) <http://www.marklogic.com/what-is-marklogic/marklogic-semantics/triple-store/>
- MemGraph (2017) <https://memgraph.com/>
- Microsoft (2017) Graph processing with SQL server and azure SQL database. <https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview>
- Neo4j (2017) <https://neo4j.com/>
- Neumann T, Weikum G (2008) RDF-3X: a RISC-style engine for RDF. *Proc VLDB Endow* 1(1): 647–659
- Ontotext (2017) <https://ontotext.com/>
- Oracle (2017a) Oracle spatial and graph. <http://www.oracle.com/technetwork/database/options/spatialandgraph/>
- Oracle (2017b) PGQL 1.1 specification. <http://pgql-lang.org/spec/1.1/>
- OrientDB (2017) <http://orientdb.com/orientdb>
- Paradies M, Lehner W, Bornhövd C (2015) GRAPHITE: an extensible graph traversal framework for relational database management systems. In: Proceedings of the international conference on scientific and statistical database management, SSDBM’15, pp 29:1–29:12
- Paradies M, Kinder C, Bross J, Fischer T, Kasperovics R, Gildhoff H (2017) GraphScript: implementing complex graph algorithms in SAP HANA. In: Proceedings of the 16th international symposium on database programming languages, DBPL’17, pp 13:1–13:4
- Raman R, van Rest O, Hong S, Wu Z, Chafi H, Banerjee J (2014) PGX.ISO: parallel and efficient in-memory engine for subgraph isomorphism. In: Second international workshop on graph data management experiences and systems, GRADES’14, pp 1–6
- RDF4J E (2017) <https://rdf4j.org/>
- van Rest O, Hong S, Kim J, Meng X, Chafi H (2016) PGQL: a property graph query language. In: Proceedings of the fourth international workshop on graph data management experiences and systems, GRADES’16, p 7
- Rodriguez MA (2015) The gremlin graph traversal machine and language. In: ACM database programming languages conference, DBPL’15
- Roth NP, Trigonakis V, Hong S, Chafi H, Potter A, Motik B, Horrocks I (2017) PGX.D/Async: a scalable distributed graph pattern matching engine. In: Proceedings of the fifth international workshop on graph data-management experiences & systems, GRADES’17, pp 7:1–7:6

- Rudolf M, Paradies M, Bornhövd C, Lehner W (2013) The graph story of the SAP HANA database. In: Datenbanksysteme für Business, Technologie und Web (BTW), BTW'13, vol 214, pp 403–420
- Saad Y (2003) Iterative methods for sparse linear systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia
- Sevenich M, Hong S, van Rest O, Wu Z, Banerjee J, Chafie H (2016) Using domain-specific languages for analytic graph databases. Proc VLDB Endow 9(13):1257–1268
- Simmen D, Schnaitter K, Davis J, He Y, Lohariwala S, Mysore A, Shenoi V, Tan M, Xiao Y (2014) Large-scale graph analytics in aster 6: bringing context to big data discovery. Proc VLDB Endow 7(13):1405–1416
- Stardog (2017) <http://www.stardog.com/>
- Stutz P, Bernstein A, Cohen W (2010) Signal/collect: graph algorithms for the (semantic) web. In: Proceedings of the 9th international semantic web conference on the semantic web – volume part I, ISWC'10. Springer, Berlin/Heidelberg, pp 764–780. <http://dl.acm.org/citation.cfm?id=1940281.1940330>
- Sun W, Fokoue A, Srinivas K, Kementsietsidis A, Hu G, Xie GT (2015) SQLGraph: an efficient relational-based property graph store. In: Proceedings of the ACM SIGMOD international conference on management of data, pp 1887–1901
- Tanase I, Xia Y, Nai L, Liu Y, Tan W, Crawford J, Lin CY (2014) A highly efficient runtime and graph library for large scale graph analytics. In: Proceedings of workshop on graph data management experiences and systems, GRADES'14, pp 1–6
- TigerGraph (2017) <https://www.tigergraph.com/>
- Valstar LD, Fletcher GHL, Yoshida Y (2017) Landmark indexing for evaluation of label-constrained reachability queries. In: Proceedings of the 2017 ACM international conference on management of data, pp 345–358
- W3C (2013) SPARQL 1.1 query language. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>

Graph Data Models

Claudio Gutiérrez¹, Jan Hidders², and Peter T. Wood³

¹Department of Computer Science and Millennium Institute on Foundations of Data, Universidad de Chile, Santiago, Chile

²Vrije Universiteit Brussel, Brussels, Belgium
³Birkbeck, University of London, London, UK

Synonyms

Graph databases; Linked data models; Network databases; Network data models

Definitions

Following the classic definition of Codd, a data model comprises three basic components: the data structure(s), a transformation and query language, and integrity constraints. Under this conceptualization, a graph data model is characterized as follows:

- The data (and possibly its schema) is represented by graphs or by generalizations of the concept of a graph (e.g., hypergraphs, hypernodes).
- The manipulation of data is done by graph transformations or by operations capturing features such as paths, neighborhoods, graph patterns, etc.
- The integrity constraints enforce the consistency of schemas and graph properties that are relevant to the particular model.

In this entry, we will focus on the data structures part, since query languages for graphs are treated in other entries in this work. Thus we will use the notion “graph data model” for the data structure of a graph data model.

Overview

Graphs have been used for centuries as a modeling tool, usually when there is a need to model the connections or relationships among a number of entities or objects. For example, the objects might be cities with the connections representing roads, or the entities might be people with the relationships being those found in families. More recently, graphs have been used to model various types of big data, such as that comprising social networks or various biological applications.

Although the notion of graph is an old one, the use of graphs in real-life modeling is rather recent. Their first uses were in knowledge representation (e.g., semantic networks Lehmann and Rodin 1992), semantic modeling (e.g., the ER model Chen 1976), and object-oriented languages (e.g., Gyssens et al. 1990 and Levene and Poulovassilis 1991).

It was not until the 1980s that the need arose to design graph data models for databases which gave rise to a “Golden Age” or “Classic Period” of graph databases that produced manifold models. The motivations were also varied, among them: representing the semantics of databases, providing natural database interfaces, generalizing the hierarchical and network models, representing knowledge structures, dealing with object-oriented features, providing transparent representation to the manipulation of graph data, and modeling hypertext (see Angles and Gutierrez 2008 for a detailed survey of this period).

These models did not reach mainstream users. It was not until the early twenty-first century that actual applications of graph databases began in practice, mainly due to huge datasets representing data in the form of graphs (e.g., social networks, complex networks, biological networks, etc.). The need to have data and its relationships directly accessible in its graph representation as well as a query language to systematically retrieve data motivated the design of several commercial systems.

We will concentrate on the three main models that are becoming both the most popular and the most studied:

1. The RDF model. Designed to be a universal representation language for resources, its basic structure consists of triples of resources of the form (s, p, o) representing the subject (s), the predicate (p), and the object (o) of a statement about these resources. Each statement (a, p, b) can be viewed as a two-node labeled graph $a \xrightarrow{p} b$. Thus, a set of RDF triples is a “graph” (even though mathematically such sets of triples are not strictly graphs, e.g., the two statements $a \xrightarrow{p} b$ and $a \xrightarrow{q} p$ are allowed.)
2. The property graph model (e.g., in Neo4j, etc.). This is a more standard model that mathematically is a graph with labels on the nodes and edges and usually multiple properties associated with them. There are several variations on the basic model (identifiers for nodes, multiple labels, etc.).

3. The nested model. This is a revival of the classic OEM (Object Exchange Model) (Papakonstantinou et al. 1995), whose most popular representation today is JSON. Each object is a set of key-value pairs, in which values can again be JSON objects. This gives a graph structure that is enriched in multiple ways (Bourhis et al. 2017).

In this entry, we will study each of these models. Earlier surveys include that by Angles and Gutierrez (2008), while surveys on query languages for graph data models, which also define the graph models under consideration, include those by Wood (2009), Barceló Baeza (2013), and Angles et al. (2017).

G

Key Research Findings

The simplest form of graph data model is that which was proposed centuries ago. A *graph* comprises a finite set of *nodes* and a finite set of *edges*, where each edge connects a pair of nodes. Graphs can be either directed or undirected, depending on whether the relationship between the pairs of nodes is symmetric or not.

More formally, an *undirected graph* G is a pair (N, E) , where N is a finite set of nodes and E is a finite set of edges, where each edge is of the form $\{u, v\}$ for $u, v \in N$. A *directed graph* G is defined similarly, except that each edge is an ordered pair of the form (u, v) for $u, v \in N$. In other words, $E \subseteq N \times N$.

Given that there is only one “type” of edge in directed and undirected graphs, their ability to model relationships is rather limited. As a result, it is common to add *labels* to the edges. These labels can then be used to denote different relationships, such as friend-of, cousin-of, etc. Since directed graphs are more general than undirected ones (an undirected graph can always be modeled by a directed one in which there is a pair of directed edges between any pair of nodes connected by an undirected edge), we will only consider directed graphs from now on.

An *edge-labeled graph* G is a triple (N, E, Lab) , where N is a finite set of nodes, E

is a finite set of edges, and Lab is a set of labels. Each edge is of the form (u, l, v) for $u, v \in N$ and $l \in Lab$; in other words, $E \subseteq N \times Lab \times N$. Edge-labeled graphs can be used to model the data comprising an extensive number of domains, and most current models use them. A good example is the Resource Description Framework (RDF).

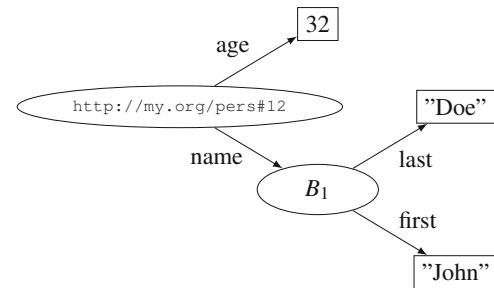
While edge-labeled graphs provide a useful way to model many kinds of data, they still have their limitations: each edge only has one piece of information associated with it, namely, its label, and each node similarly has a single identifier. Sometimes it is important to be able to add more information about an edge, for example, who added it to the graph or for how long it is valid. Nodes too may need to have a number of properties associated with them, especially if they correspond to real-world objects. These requirements have given rise to what have been called property graphs.

The Resource Description Framework

The Resource Description Framework (RDF) is a W3C recommendation (Cyganiak et al. 2014), originally designed to represent metadata on the web but more recently used as a representation layer for the so-called semantic web (Berners-Lee et al. 2001). As mentioned above, an RDF graph can be viewed as an example of an edge-labeled graph, that is, although an RDF graph is a set of triples of the form (u, l, v) , where u , l and v are universal resources, one can view u and v as nodes and l as an edge label of a graph.

An atomic RDF expression is a triple comprising a *subject* u (the resource being described), a *predicate* l (the property of the resource being described), and an *object* v (the value of the property). Known resources are identified by IRIs (Internationalized Resource Identifiers), while unknown resources are denoted by so-called blank nodes. This set can be partitioned into disjoint sets of IRIs, blank nodes, and literals. However, in RDF further constraints are applied to the permitted sets of nodes, labels, and triples.

Example 1 (Simple person example) A basic RDF graph describing John Doe and his age is the following: $G = \{(I_1, \text{name}, B_1), (B_1, \text{first}, \text{"John"}), (B_1, \text{last}, \text{"Doe"}), (B_1, \text{age}, 32)\}$, where $I_1 = \text{http://my.org/pers\#12}$, which graphically looks as follows:



In G , the node with content <http://my.org/pers#12> is a known resource identified by an IRI. The node B_1 is a blank node as mentioned above. The predicates (name, first, last, age) are part of the schema (ontology). Blank nodes and predicates have also IRIs (not shown to save space). The rest are value types (strings and integers in this case). Any resource (IRI) can be further referenced in a triple in another graph, in this way integrating different graphs.

As mentioned, some distinguished resources conform to schemas. A schema in RDF is simply a reserved vocabulary, usually further specified, which can be used to describe other resources. Examples above are “name,” “age,” etc. The RDF specification includes a basic vocabulary (RDF schema) consisting of type, class, property, subClass, subProperty, and the like, plus some basic deductive rules, e.g., transitivity of subClass, etc. (Brickley and Guha 2014).

Although the original idea of RDF was that of a metadata specification for the web, it became a standard to build graph databases with RDF data. In this direction, a standard query language, called SPARQL (Harris and Seaborne 2013), was introduced to query RDF data.

The Property Graph Model

A *property graph* G is a directed multigraph with possibly multiple labels on nodes and edges. Formally, G is a tuple $(N, E, Lab, Prop, Val, \rho, \lambda, \sigma)$, where N and E are finite sets of nodes and edges; $Prop$ is a finite set of properties; Lab and Val are sets of labels and values, respectively;

and ρ , λ , and σ are functions defined as follows (Angles et al. 2017):

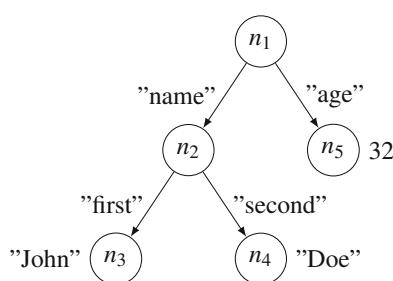
- $\rho : E \rightarrow (N \times N)$ is a total function defining the edges, where, for $e \in E$, $\rho(e) = (u, v)$ means that e is a directed edge from u to v for $u, v \in N$.
- $\lambda : (N \cup E) \rightarrow Lab$ is a total function providing the label for each node and edge. If $v \in N$ and $e \in E$, then $\lambda(v)$ is the label of node v and $\lambda(e)$ is the label of edge e .
- $\sigma : (N \cup E) \times Prop \rightarrow Val$ is a partial function providing values for those properties defined for particular nodes and edges. If $v \in N$, $e \in E$ and $p \in Prop$, then $\sigma(v, p)$ is the value for property p for node v , if defined (and similarly for edge e).

Example 2 Consider again the simple person example used in Example 1. This can be represented as a property graph as follows:

- Nodes $N = \{n_1, n_2, n_3, n_4, n_5\}$.
- Edges $E = \{e_1 = (n_1, n_2), e_2 = (n_1, n_5), e_3 = (n_2, n_3), e_4 = (n_2, n_4)\}$.
- Node labels $\lambda(n_1) = (), \lambda(n_2) = (), \lambda(n_3) = "John", \lambda(n_4) = "Doe", \lambda(n_5) = 32$.
- Edge labels $\lambda(e_1) = "name", \lambda(e_2) = "age", \lambda(e_3) = "first", \lambda(e_4) = "second"$.

Here we have identifiers for all nodes and edges. The rest – strings and numbers – are values, including $()$, which represents the absence of any value. Only nodes can be further referenced. Edges have identifiers only for the purpose of attaching properties to them.

Graphically, the example can be represented as follows:



The schema in current property graph databases plays essentially the role of integrity constraints to ensure safe modeling or improve performance. So a schema includes the possible properties for nodes and edges, the corresponding types, possible checking for existence or cardinality, etc. Some systems include indexes as part of the schema.

Current property graph database systems include storage, a query language (although there is no widely adopted standard), APIs, and application functionalities such as analytics, visualization, and tools to produce code or assemble with other applications. The Neo4J Graph Platform is a native graph database supporting transactional applications and graph analytics. It has developed its own query language called Cypher (<https://neo4j.com>). JanusGraph is designed for storing and querying graphs distributed across a multi-machine cluster. It uses Gremlin as its query language (<http://janusgraph.org/>). DataStax Enterprise Graph is part of the DataStax Enterprise (DSE) data platform. It is built on Apache Cassandra (<http://cassandra.apache.org>) and also uses the Gremlin query language. It supports distribution and benefits from the other functionalities of DSE (<https://www.datastax.com/products/datastax-enterprise-graph>).

The Nested Data Model

The need for a data model that was more flexible than the relational data model originally arose when trying to integrate heterogeneous data into a single representation. This gave rise to the Object Exchange Model (OEM) and subsequently alternative representations for what was to be termed *semi-structured data*. The most recent concrete proposal in this area is JSON (JavaScript Object Notation).

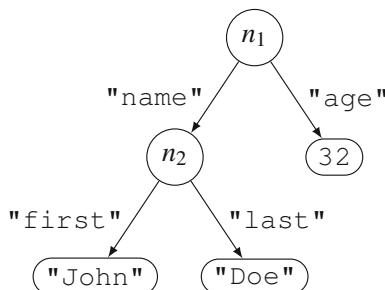
JSON is a lightweight representation based on data types used in JavaScript and is useful when exchanging data between applications. Apart from the primitive values of true, false, and null, and the primitive types of strings and numbers, JSON provides two structuring types, namely, objects and arrays. Objects are represented by key-value pairs (dictionaries), while arrays are simply sequences of values. The values appearing in objects and arrays can be arbitrary

JSON constructs, thereby allowing arbitrary nesting of structures.

Example 3 Consider once again the person example from Example 1. This can be represented in JSON as follows:

```
{
  "name": {
    "first": "John",
    "last": "Doe"
  },
  "age": 32
}
```

The basic structure in JSON is the key-value pair, with key-value pairs being grouped into objects, each delimited by braces. Each key within an object must be a string and must also be unique. Each value can be a basic value (number, string, etc.) or a nested expression. The underlying structure is a graph, in this case:



A schema language for describing what can be expected in a JSON document is JSON Schema (<http://json-schema.org/>), a draft proposal for an IETF standard. It allows the definition of types for JSON values and JSON documents. A type specification usually indicates the basic type, which might be *object*, *array*, or one of the literal types such as *numeral*, *string*, or *null*.

Type definitions in JSON Schema can be assigned an identifier, which can be used to let type definitions recursively refer to each other. This makes it possible to have recursively nested types, but the recursion is restricted to ensure well-defined semantics.

Several DBMSs support JSON as a data format. Those that use it as their main data format are also referred to as *document stores*, *document-oriented databases*, or *JSON databases*. The most well-known of these include *MongoDB* (<https://www.mongodb.com>), *CouchDB* (<http://couchdb.apache.org>), and *Azure Cosmos DB* (<https://azure.microsoft.com/en-gb/services/cosmos-db>). For example, *MongoDB* offers a schema language (its own and JSON Schema) and a proprietary query language to select and query JSON documents. Along with these document stores, there are NoSQL databases, such as *MarkLogic* (<http://www.marklogic.com>) and *OrientDB* (<http://orientdb.com>) that support JSON as one of their data formats. Finally, many popular SQL DBMSs, such as *MySQL* (<https://www.mysql.com>), *PostgreSQL* (<https://www.postgresql.org>), and *Oracle* (<https://www.oracle.com/database/index.html>), support JSON.

A Brief Comparison of Models

Graph data models are designed to represent data and their relationships in an explicit form. Traditionally in mathematics, graphs are defined by a *universe*, the set of nodes, and over it, the edges are defined as an additional structure, either as pairs (directed edges) or two-element sets (undirected edges), allowing one edge or multiple edges between two nodes, etc.

In all three models presented, the two defining sets of elements (nodes and edges) have their own identifiers, that is, are objects themselves. (This marks the difference between the graph model and, say, the relational model: in the latter, only nodes have identifiers, and edges are defined as relations or “tuples” of nodes.) In computer science, this is represented by a two-typed model: vertices and edges. Over this basic level, one can enrich vertices and/or edges. The three models considered do this in different directions.

JSON keeps edges as basic pairs of nodes but enriches the notion of node by allowing recursive structures for them. This allows “edges” between “higher-level” nodes that, once flattened, give a

graph structure. Properties in the form of labels are described inside these structures.

The RDF model keeps the nodes as primitives and enriches the edges with a label that can be a node. This label allows the structure to be enriched. But note that edges do not properly have identifiers (they are pairs of nodes). One can get RDF “graphs” such as $(a, b, c), (b, p, d)$ and (d, q, c) that can be represented as $\{a \xrightarrow{b} c, b \xrightarrow{p} d, d \xrightarrow{q} c\}$. This allows great flexibility, by allowing to “describe” the edge label b recursively. The price for this flexibility is the complexity to implement the model efficiently (because there is a mix between properties of objects and the objects themselves).

The property model keeps identifiers for nodes and edges and enriches the structure by adding additional properties over them. These properties do not have (as in the nested and RDF models) any intersection with the primitive nodes and edges. This model keeps the types (nodes, edges, properties) clearly disjoint, and thus classical graph algorithms can be implemented directly.

Conclusions

In this entry we have concentrated on only three categories of graph data model: edge-labeled graphs as exemplified by RDF, the property graph model, and the nested graph model. We believe these are currently the graphs models that are most widely studied and implemented. Many other graph data models are described in the survey by Angles and Gutierrez (2008).

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Query Languages](#)
- ▶ [Linked Data Management](#)

References

- Angles R, Gutierrez C (2008) Survey of graph database models. *ACM Comput Surv* 40(1):1:1–1:39
- Angles R, Arenas M, Barceló P, Hogan A, Reutter JL, Vrgoč D (2017) Foundations of modern

graph query languages. *ACM Comput Surv* 50(5):68:1–68:40

Barceló Baeza P (2013) Querying graph databases. In: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*. ACM, New York, pp 175–188

Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. *Sci Am* 284(5):34–43

Bourhis P, Reutter JL, Suárez F, Vrgoč D (2017) JSON: data model, query languages and schema specification. In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*. ACM, New York, pp 123–135

Brickley D, Guha R (2014) RDF schema 1.1. <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>, W3C Recommendation 25 Feb 2014

Chen PPS (1976) The entity-relationship model – toward a unified view of data. *ACM Trans Database Syst* 1(1):9–36

Cyganiak R, Wood D, Lanthaler M (2014) RDF 1.1 concepts and abstract syntax. <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>, W3C Recommendation 25 Feb 2014

Gyssens M, Paradaens J, Den Bussche JV, Gucht D (1990) A graph-oriented object database. In: *Proceedings of the 9th symposium on principles of database systems*. ACM Press, pp 417–424

Harris S, Seaborne A (2013) SPARQL 1.1 query language. <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>, W3C Recommendation 21 Mar 2013

Lehmann FW, Rodin EY (eds) (1992) Semantic networks in artificial intelligence. International series in modern applied mathematics and computer science, vol 24. Pergamon Press, Oxford

Levene M, Poulovassilis P (1991) An object-oriented data model formalised through hypergraphs. *IEEE Trans Knowl Data Eng* 6(3):205–224

Papakonstantinou Y, Garcia-Molina H, Widom J (1995) Object exchange across heterogeneous information sources. In: *Proceedings of the eleventh international conference on data engineering*. IEEE, p 251–260

Wood PT (2009) Graph database. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, Boston, pp 1263–1266

Graph Databases

- ▶ [Graph Data Models](#)

Graph Drawing

- ▶ [Graph Visualization](#)

Graph Exploration and Search

Davide Mottin¹ and Yinghui Wu²

¹Hasso Plattner Institute, Potsdam, Germany

²Washington State University, Pullman, WA, USA

Definitions

Exploratory methods have been proposed as a mean to extract knowledge from relational data without knowing what to search (Idreos et al. 2015). *Graph exploration* has been introduced to perform exploratory analyses on graph-shaped data (Mottin and Müller 2017). Graph exploration aims at mitigating the access to the data to the user, even if such user is a novice.

Algorithms for graph exploration assume the user is not able to completely specify the object of interest with a structured query like a SPARQL (see chapter “► [Graph Query Languages](#)”), but rather expresses the need with a simpler, more ambiguous language.

This asymmetry between the rigidity of structured queries and ambiguity of the user has inspired the study of approximate, flexible, and example-based methods.

Overview

The research on graph exploration has revolved around three main pillars: *keyword graph queries*, *exploratory graph analysis*, and *refinement of query results*.

Keyword graph queries applies user-friendly keyword queries to explore a targeted graph. The query answers are usually defined as subgraphs that contain the nodes and edges with information specified by the query terms. New keywords can be suggested from observed answers and are used to expand the original queries, which triggers a new round of query evaluation. Keyword queries are desirable for end users who do not have prior knowledge of underlying graph data, for which

writing a complex graph query (e.g., SPARQL) is difficult. On the other hand, keywords can be ambiguous. While keyword search has been extensively studied, the major research effort in this category focuses on disambiguation and keyword query refinement.

Exploratory graph analysis entails the process of casting an incomplete or imperfect pattern query to let the system find the closest match. Such exploratory analysis may return a huge number of results, e.g., structures matching the pattern. Thus, the system is required to provide intelligent support. One such strategy is the well-known query-by-example paradigm, in which the user provides the template for the tuples and lets the system infer the others.

Refinement of graph query results is needed to deal with the overwhelming amount of results that is typical in subgraph processing. It includes approaches designed to present comprehensive result sets to the user or intermediate results that can be refined further.

Key Research Findings

Graph Search with Keywords

Keyword search (KWS) has been routinely used to explore graph data (Wang and Aggarwal 2010; Mottin and Müller 2017). (1) A keyword query Q is a set of terms $\{t_1, \dots, t_n\}$. Given graph $G=(V, E)$ and a term t_i , a match function determines a set of *content nodes* $V(t_i) \subseteq V$ that match t_i . (2) An answer G_Q of a keyword query Q is a subgraph of G that contains a node from $V(t_i)$ for each $t_i \in Q$. A function $F(G_Q)$ is used to quantify the cost of connecting all content nodes. Answers with smaller cost have higher quality.

Given a node pair (u, v) in graph G , (1) the distance from u to v , denoted as $\text{dist}(u, v)$, is the sum of edge weight on the shortest path from u to v ; and (2) $\text{len}(u, v)$ denotes the length of the shortest path from u to v . There are three common classes of KWS queries.

Distinct root-based KWS defines G_Q as a minimal rooted tree which (1) contains a distinct root node v_r and at least a content node $v_i \in V(t_i)$ as a leaf for each $t_i \in Q$ and (2) $\text{len}(v_r, v_i) \leq r$ for a predefined hop bound r , for each leaf v_i . Here, G_Q is minimal if no subtree of G_Q is an answer of Q in G . The function $F(G_Q)$ is defined as $F(G_Q) = \sum_{t_i \in Q} \text{dist}(v_r, v_i)$, where v_i ranges over the content nodes. The answers of such queries can be found in $O(|Q|(|V| \log |V| + |E|))$ time (cf. Yu et al. 2010).

Steiner tree-based KWS (Bhalotia et al. 2002) differs from its distinct root-based counterparts in that it uses a different cost function $F(G_Q)$, which is defined as $\sum_{e \in G_Q} w(e)$, i.e., the total weight of the edges in the Steiner tree G_Q . It is NP-hard (Yu et al. 2010) to evaluate a Steiner tree-based query by computing a minimum weighted Steiner tree, a known NP-hard problem (Ding et al. 2007). Both exact (Ding et al. 2007) and approximation algorithms (Byrka et al. 2013) have been developed to evaluate such queries.

Steiner graph-based KWS (Li et al. 2008; Kargar and An 2011) finds G_Q as graphs rather than trees. For Steiner graph-based queries with a number r , G_Q is a Steiner graph that contains content and Steiner nodes (i.e., nodes on shortest paths between two content nodes), with either radius bounded by r (i.e., r -radius Steiner graph (Li et al. 2008)) or distance between any two content nodes bounded by r (i.e., r -clique (Kargar and An 2011)). For an answer G_Q with nodes $\{v_1, \dots, v_n\}$, its cost $F(G_Q)$ is computed as $\sum_{i \in [1, n]} \sum_{j \in [i+1, n]} \text{dist}(v_i, v_j)$, i.e., the total pairwise distances of the content nodes in G_Q . It is in general NP-hard to evaluate Steiner graph-based queries (Yu et al. 2010). Approximate algorithms are developed for such queries to find r -radius graphs (Li et al. 2008) and r -cliques (Kargar and An 2011).

Keyword query suggestion. Keyword query suggestion and its variants (e.g., query refinement and reformulation) have been studied to suggest new queries that better describe search intent

for graph exploration. Most prior work adopts information retrieval techniques that make use of query logs and user feedback (Carpinetto and Romano 2012). Keyword query suggestion for graphs has been studied in Tran et al. (2009) by suggesting structured queries computed from the keywords over RDF data. In order to summarize the results of KWS over structured data, tag clouds (Koutrika et al. 2009) discover the most significant words retrieved as a part of the initial results. Provable quality guarantees of the answers are not addressed in these works.

Query suggestion has been studied for XML data (Zeng et al. 2014), with a focus on coping with missed matches. An approximate answer of original query Q is computed by expanding its answer with content nodes of the same type, and new keywords are suggested to replace those without match in Q .

Exploratory Graph Analysis

One of the earliest attempts employing examples as search conditions is query-by-example (Zloof 1975). The main idea was to help the user in the query formulation, allowing her to specify the shape of the results in terms of templates for tuples, i.e., examples. Query by example has been lately revisited, and the use of examples has found application in graph data. The definition of example has transformed from a mere template to the representative of the intended results the user would like to retrieve. *Examples* for a graph $G = (V, E)$ can be either *set of nodes* $Q \subseteq V$, *tuple* of nodes $(v_1, \dots, v_n) \in V^n$, or *subgraphs* $Q \subseteq G$, where with an abuse of notation \subseteq indicates both set and structural inclusion.

Set of nodes as examples. Several analyses can be performed when a set of example nodes $Q \subseteq V$ is provided; this includes the discovery of dense graph regions (Gionis et al. 2015; Ruchansky et al. 2015), central nodes (Tong and Faloutsos 2006), and communities (Staudt et al. 2014; Perozzi et al. 2014). Discrepancy maximization in graphs (Gionis et al. 2015) aims at finding a connected subgraph $G' = (V', E')$, $G' \subseteq G$ that maximizes the discrepancy $\delta(G') = \alpha|Q| - |V \setminus Q|$, $\alpha > 1$; in other words,

G' is a subgraph containing more nodes from Q than other nodes. Discrepancy maximization is NP-hard. On a similar line, the centerpiece subgraph problem (CEPS) (Tong and Faloutsos 2006) finds the maximum weight subgraph that contains at least k query nodes and at most b other nodes. A generalization of the CEPS problem, the minimum Wiener connector problem (MWC) (Ruchansky et al. 2015) computes dense regions of a graph potentially representing community structures, given a set of nodes as input. MWC aims at finding a subgraph of G induced by Q , denoted as $G[Q]$, that minimizes the sum of the pairwise distances among the nodes in Q , i.e., $\arg \min_{G[Q]} \sum_{\{u,v\} \in Q} d_{G[Q]}(u, v)$.

Examples have also been used as a mean to detect communities, where a community is a set of nodes, in which every node is connected to many more nodes inside the community than outside. One approach is to consider each node $v \in Q$ as a representative of a different community (Staudt et al. 2014). In graphs with attributes on nodes, FocusCO (Perozzi et al. 2014) introduced a clustering algorithm that learns a similarity function among nodes, based on the attributes in the examples. Formally, FocusCO takes an attributed graph $G = (V, E, F)$, where F is a feature matrix $F \in \mathbb{R}^n \times f$ and F_i represents the attribute vector for node v_i , and returns clusters that better represent the seed nodes Q . FocusCO returns outliers that do not belong to any community identified so far. (For community detection, see chapter “Community Analytics in Graphs.”)

Node tuples as examples. In graph query by example (GQBE) (Jayaram et al. 2015b) a tuple of nodes $(v_1, \dots, v_n), v_i \in V^n$ is a representative of the result tuples. For instance, if the pair (*Yahoo, Jerry Yang*) is provided, the result is expectedly other *IT company, founder* pairs. The solution first computes the *minimal answer graph* connecting the query nodes and then finds other subgraphs isomorphic to such graph.

Subgraphs as examples. Subgraphs are more expressive than nodes and tuples; hence they can

be exploited to obtain more accurate results. In the case of graphs, the example can constitute an exemplar query (Mottin et al. 2016) $Q \subseteq G$, and a result is a subset of G congruent to Q . Exemplar queries is a flexible paradigm that allows the definition of multiple congruence relations among the input example and the intended results, such as isomorphism or strong simulation (Ma et al. 2014). Moreover, it supports efficient retrieval of top- k results.

A generalization of exemplar queries, PANDA (Xie et al. 2017), studies partial topology-based network search, that is, finding the connections (paths) between structures node-label isomorphic to different user inputs. PANDA first materializes all isomorphic graphs, then groups them into connected components, and finally finds undirected shortest paths among them. More techniques on graph pattern queries are introduced in chapters “► Graph Pattern Matching” and “► Graph Query Processing.”

Reformulation of Graph Queries

Users with exploratory intent typically provide indefinite queries that are likely to return either too few or too many results. For this reason, query reformulation techniques aim at modifying the query to lead the user to the intended result (Motin et al. 2015; Hurtado et al. 2008; Islam et al. 2015). A *reformulation* for a query $Q \subseteq G$ on a labeled graph $G = (V, E, \ell)$ with labeling function on nodes and edges ℓ is another query Q' that is either a supergraph $Q' \supseteq Q$ or a subgraph $Q' \subseteq Q$ of Q . Therefore, Q' can be more specific (supergraph) or more generic (subgraph) than Q . Queries considered in graph query reformulation are usually subgraph isomorphism queries (Lee et al. 2012). Query reformulation has been proposed for collections of graphs, or *graph databases*, as well as *large graphs*.

Query reformulation in graph databases. Query reformulation in graphs can apply to collection of graphs or *graph databases* $\mathcal{D} = \{G_1, \dots, G_n\}$, where each G_i is a graph $G_i = (V_i, E_i, \ell_i)$. In such graph databases, the query Q returns a subset of the collection $\mathcal{R}(Q) \subseteq \mathcal{D}$ containing the structure Q .

Graph query reformulation (GQRef) (Mottin et al. 2015) discovers query reformulations having small overlap and high coverage; thus ideally no result is missing from the reformulated queries, yet the reformulations have little redundancy. The objective is to find a set R of k reformulations with high coverage and diversity (i.e., small overlap) maximizing $f(R) = |\bigcup_{Q \in R} \mathcal{R}(Q)| + \lambda \sum_{Q_1, Q_2 \in R} |\mathcal{R}(Q_1) \cup \mathcal{R}(Q_2)|$, with λ trading-off coverage and diversity. Maximizing f is NP-hard, but an approximate greedy solution can guarantee a $(1/2)$ -approximation.

Alternatively, AutoG (Yi et al. 2017) proposes a top- k reformulation approach for visual autocomplete of queries. AutoG returns the k best relaxations according to a ranking function that favors a large number of results for each reformulation and large diversity. Such ranking function (excluding normalization) is $u(R) = \alpha \sum_{Q \in R} |\mathcal{R}(Q)| / |\mathcal{D}| + (1 - \alpha) \sum_{Q_1, Q_2 \in R} \text{dist}(Q_1, Q_2)$, where $\alpha \in [0, 1]$ strikes a balance between coverage and a user-defined distance dist among queries. For visual query analysis, see chapter “► [Visual Graph Querying](#).”

Query reformulation in large graphs. Few works explicitly explore query reformulation in large graphs. Early attempts in this direction propose to enrich the SPARQL semantics with a RELAX clause that returns a set of reformulations following some predefined rules (such as expanding edges with a specific label) (Hurtado et al. 2008). In Arenas et al. (2014) the SPARQL reformulations are extended to OWL logic, where the semantics of each reformulation is captured by the RDF taxonomy. The user can interactively accept or reject the possible expansions of a query such that the navigation is consistent with the rules.

Query reformulation is also employed in query debugging to understand why the query returned too-many or too-few answers (Vasilyeva et al. 2016). In order to explain the dependency between the number of results and the query (Vasilyeva et al. 2016), first compute the *maximum common subgraph* (MCS) among the graph G

and the query Q and then use a *differential graph* to profile the query by adding edges from the differential graph to the MCS. The maximum common subgraph is the largest connected graph G' that is subgraph of both Q and G .

VIIQ (Jayaram et al. 2015a) employs query logs to rank the possible reformulations of the query Q . VIIQ’s ranking function generates *correlation paths* among the sequence of reformulations in the current query Q and the sequences in the query log. Among the candidate *single* edges that can be added to Q , the ranking function proposes those that have a larger support in the query log.

Why-not queries in graphs. Why-not queries additionally require the user to provide a set of missing answers from the results of Q , and the system proposes a new query Q' returning those results and the least number of irrelevant results. The only work in this direction is Islam et al. (2015). In order to compute why-not queries, the maximum common subgraph (MCS) among the query Q and the missing answers \mathcal{D}^- is computed. The best query is found minimizing the maximum distance among Q' and the missing answers \mathcal{D}^- as $\arg \min_{Q' \in \mathcal{D}} \max\{\Delta(Q', G) | G \in D^-\}$, where given $G_1 = (E_1, V_1, \ell_1)$, $G_2 = (E_2, V_2, \ell_2)$, $\Delta(G_1, G_2) = |E_1| + |E_2| - 2|\text{MCS}(G_1, G_2)|$.

G

Key Applications

The need of graph exploratory is evident in user-friendly knowledge exploration (Achiezra et al. 2010), “why-not” query processing (Tran and Chan 2010), and database security, where highly correlated queries are identified to reverse engineering the sensitive entity information (Tran et al. 2014). Another application is interactive sensemaking of complex graph data (Pienta et al. 2015), with broader applications in literature search and collaboration recommendation in academic networks, attack and anomaly detection in cyber networks, and event detection in social media.

Future Directions

The challenges remain open for interactive graph exploration. Real-time interactivity is crucial to exploration over large graphs. This requires scalable graph analytics and query suggestion; both are nontrivial for graph data. One direction can be to integrate emerging scalable graph mining and learning methods to graph exploration tasks, beyond query-based exploration. Uncertainty and trustworthiness of graph data sources is another challenge for exploration tasks, especially for knowledge bases. This requires truth-aware exploration that guarantees the correctness of retrieved information. A third direction is to combine usable user interfaces and scalable graph visualization techniques to facilitate sensemaking and knowledge discovery by graph exploration.

Cross-References

- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)
- ▶ [Graph Query Processing](#)
- ▶ [Visual Graph Querying](#)

References

- Achiezra H, Golenberg K, Kimelfeld B, Sagiv Y (2010) Exploratory keyword search on data graphs. In: SIGMOD, pp 1163–1166
- Arenas M, Cuenca Grau B, Kharlamov E, Marcuska S, Zheleznyakov D (2014) Faceted search over ontology-enhanced RDF data. In: CIKM. ACM, pp 939–948
- Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S (2002) Keyword searching and browsing in databases using banks. In: ICDE, pp 431–440
- Byrka J, Grandoni F, Rothvoss T, Sanità L (2013) Steiner tree approximation via iterative randomized rounding. *J ACM* 60(1):6
- Carpinetto C, Romano G (2012) A survey of automatic query expansion in information retrieval. *ACM Comput Surv (CSUR)* 44:1
- Ding B, Yu JX, Wang S, Qin L, Zhang X, Lin X (2007) Finding top-k min-cost connected trees in databases. In: ICDE, pp 836–845
- Gionis A, Mathioudakis M, Ukkonen A (2015) Bump hunting in the dark – local discrepancy maximization on graphs. In: ICDE
- Hurtado CA, Poulovassilis A, Wood PT (2008) Query relaxation in RDF. *J Data Semant* 4900(Chapter 2): 31–61
- Idreos S, Papaemmanoil O, Chaudhuri S (2015) Overview of data exploration techniques. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 277–281
- Islam MS, Liu C, Li J (2015) Efficient answering of why-not questions in similar graph matching. *TKDE* 27(10):2672–2686
- Jayaram N, Goyal S, Li C (2015a) VIIQ: auto-suggestion enabled visual interface for interactive graph query formulation. *PVLDB* 8(12):1940–1943
- Jayaram N, Khan A, Li C, Yan X, Elmasri R (2015b) Querying knowledge graphs by example entity tuples. *IEEE Trans Knowl Data Eng* 27(10):2797–2811. <https://doi.org/10.1109/TKDE.2015.2426696>
- Kargar M, An A (2011) Keyword search in graphs: finding r-cliques. In: VLDB, pp 681–692
- Koutrika G, Zadeh ZM, Garcia-Molina H (2009) Data clouds: summarizing keyword search results over structured data. In: EDBT, pp 391–402
- Lee J, Han WS, Kasperovics R, Lee JH (2012) An in-depth comparison of subgraph isomorphism algorithms in graph databases. In: PVLDB, VLDB Endowment, vol 6, pp 133–144
- Li G, Ooi BC, Feng J, Wang J, Zhou L (2008) Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp 903–914
- Ma S, Cao Y, Fan W, Huai J, Wo T (2014) Strong simulation: capturing topology in graph pattern matching. *ACM Trans Database Syst (TODS)* 39(1):4
- Mottin D, Müller E (2017) Graph exploration: from users to large graphs. In: SIGMOD, pp 1737–1740
- Mottin D, Bonchi F, Gullo F (2015) Graph query reformulation with diversity. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 825–834
- Mottin D, Lissandrini M, Velegakis Y, Palpanas T (2016) Exemplar queries: a new way of searching. *VLDJB* 25(6):741–765
- Perozzi B, Akoglu L, Iglesias Sánchez P, Müller E (2014) Focused clustering and outlier detection in large attributed graphs. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1346–1355
- Pienta R, Abello J, Kahng M, Chau DH (2015) Scalable graph exploration and visualization: sensemaking challenges and opportunities. In: 2015 International Conference on Big Data and smart computing (BigComp), pp 271–278
- Ruchansky N, Bonchi F, García-Soriano D, Gullo F, Kourtellis N (2015) The minimum wiener connector problem. In: SIGMOD. ACM, New York, pp 1587–1602
- Staudt CL, Marrakchi Y, Meyerhenke H (2014) Detecting communities around seed nodes in complex networks. In: IEEE international conference on big data (Big Data). IEEE, pp 62–69

- Tong H, Faloutsos C (2006) Center-piece subgraphs: problem definition and fast solutions. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 404–413
- Tran QT, Chan CY (2010) How to conquer why-not questions. In: SIGMOD, pp 15–26
- Tran T, Wang H, Rudolph S, Cimiano P (2009) Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: ICDE, pp 405–416
- Tran QT, Chan CY, Parthasarathy S (2014) Query reverse engineering. In: VLDB, pp 721–746
- Vasilyeva E, Thiele M, Bornhövd C, Lehner W (2016) Answering “why empty?” and “why so many?” queries in graph databases. *J Comput Syst Sci* 82(1):3–22
- Wang H, Aggarwal CC (2010) A survey of algorithms for keyword search on graph data. In: Managing and mining graph data, pp 249–273
- Xie M, Bhowmick SS, Cong G, Wang Q (2017) Panda: toward partial topology-based search on large networks in a single machine. *VLDBJ* 26(2):203–228
- Yi P, Choi B, Bhowmick SS, Xu J (2017) Autog: a visual query autocompletion framework for graph databases. *VLDBJ* 26(3):347–372
- Yu JX, Qin L, Chang L (2010) Keyword search in relational databases: a survey. *IEEE Data Eng Bull* 33(1): 67–78
- Zeng Y, Bao Z, Ling TW, Jagadish H, Li G (2014) Breaking out of the mismatch trap. In: ICDE, pp 940–951
- Zloof MM (1975) Query by example. In: Proceedings of the May 19–22, 1975, national computer conference and exposition. ACM, pp 431–438

Graph Generation and Benchmarks

Angela Bonifati¹ and Arnau Prat-Pérez²

¹Lyon 1 University, Villeurbanne, France

²Universitat Politècnica de Catalunya, Barcelona, Spain

Definitions

Benchmarking has been crucial for the uptake and evolution of database technologies. Benchmarks allow systems to compete on a fair, non-biased setup, giving users an understanding of how two or more systems would compare in the same real setting. Additionally, the competition for obtaining the best benchmark scores has

guided the research and development of database systems during years, speeding up their progression and their impact in society. Among many benchmarking initiatives, the Transaction Processing Council (TPC 2017) family of benchmarks is the best example of influential database benchmarks.

Industry and academia are aware of the benefits benchmarking can provide to the evolution and adoption of graph database technologies, and as such, many graph benchmarking initiatives have emerged such as Erling et al. (2015), Iosup et al. (2016), Armstrong et al. (2013), or Bagan et al. (2017) just to cite a few of them. However, designing a benchmark is not a trivial task. A benchmark has several components that need to be carefully designed/selected. For instance, which is the target audience of the benchmark? Which are the datasets the benchmark works with? How do we quantitatively measure the performance of a system running the benchmark? These are just a few questions that must be answered when designing a benchmark.

In this chapter, we review the elements a graph benchmark consists of and how the different existing graph database benchmarks address them. Finally, we discuss the different open challenges in graph benchmarking.

Overview

Given the diversity of systems and use cases, graph database benchmarks are typically designed to target specific types of systems working on a reference use case. These highly influence the three main components that form a database benchmark: the *workload*, the *scoring metric*, and the *auditing rules*.

The Workload

The main component of a benchmark is the workload, which consists of the definition of the queries the *systems under test (SUTs)* will execute, the query mix (i.e., how many instances of each query and when these will be executed), and the data that will be used.

Queries and Query mixes. There are different strategies usually followed when designing the queries of a benchmark workload. One commonly adopted strategy is the one based on analyzing a reference use case (e.g., the operation of a social network web site) in depth. From this analysis, representatives of the most relevant queries (e.g., a friend recommendation query, a group recommendation query, etc.) are synthesized, which exhibit either a read or a write mode. Additionally, the query mix is carefully designed to reproduce the query frequencies typically observed. This strategy pursues to accurately model the reference use case with the goal of providing the user with insight about how different SUTs will perform in a similar but real setting. Benchmarks following this approach are typically known as *macro-benchmarks*.

However, the purpose of a benchmark is not only to assess of the current status of an industry but also to drive its evolution and path to maturity. Thus, some benchmarks do not only focus on accurately modeling the reference use case queries but also on making the queries hard to challenge the different industry and academia actors. This is usually achieved by understanding how queries are typically executed on the target systems and by identifying which low-level operations or operators dominate the execution time of typical queries issued against the SUTs and become their bottlenecks. Then, queries are crafted (in combination with the input data) to deliberately stress such low-level operations and to push the SUTs to their limits. When following this strategy, the benchmark designer must find a trade-off between the realism of the queries and their hardness. This also applies to the definition of the query mix, which may consist of frequencies that do not necessarily strictly reproduce those observed in a reference use case. For instance, one might decide to increase the frequency of particular operations (e.g., write operations) to stress the systems in this aspect. This approach has the advantage of testing the SUTs more comprehensively (bringing them out of their comfort zone), which in turn stimulates their evolution toward usages not considered by the reference use case, and making the whole industry to advance.

By pushing this strategy to an extreme, one typically ends up with a *micro-benchmark*, which are benchmarks whose queries are directly the low-level operations/operators used to build more complex queries (e.g., getting the k-hop neighborhood of a node, computing the path between two nodes, performing a join, etc.).

The Data. The performance of most of the queries in database applications is data driven, that is, it is highly influenced by the characteristics of the data used. This is especially true for graph database applications, which usually perform expensive operations. For instance, one might think about the computation of the k-hop neighborhood of a given node. Depending on the query node, its degree, and the diameter of the graph, a hypothetically small two- or three-hop neighborhood could lead to visit almost all nodes of the graph (Backstrom et al. 2012). Thus, when designing the workload, designers must carefully choose the data to be used, not only to meet functional requirements (e.g., the presence of attributes that are involved in the queries) but also other characteristics that may affect the performance of the SUTs (number of joins, number of recursive steps, number of disjuncts, and so on).

In general, benchmarks use two types of datasets: real datasets gathered from the real reference use case and synthetically generated datasets. The former has the advantage of containing the desired characteristics, but they are typically scarce since their owners are reluctant to share them, and they do not always obey the desired scale. For this reason, many benchmark designers opt for using synthetic graph generators, paying attention to accurately model the characteristics observed in their realistic counterparts. This strategy has the advantage that graphs at different scales can be modeled, allowing the definition of *scale factors* to test the SUTs at different scales. The importance of synthetic graph generation has significantly grown during the last years, and thus there is a considerable amount of research on the topic.

Real networks like social networks, biological networks, scientific collaboration networks, or the World Wide Web share particular characteristics including a skewed degree distribution that can be approximated with a power law with a long tail, a community structure, a small diameter, or a large largest connected component (Girvan and Newman 2002; Newman 2003; Boccaletti et al. 2006). As such, many synthetic graph generation techniques focus on reproducing such observed characteristics, including the method proposed by Newman et al. (2001), the RMAT method proposed by Chakrabarti et al. (2004), the Kronecker graphs proposed by Leskovec et al. (2005), the LFR graph generator introduced by Lancichinetti et al. (2008), or the BTER graph generator described by Kolda et al. (2014), just to cite a few of them. Typically, graph benchmarks adopt these techniques to build their own graph generators, since the former are mostly focused on the structure of the graph but do not consider the generation of properties, usually required by benchmarks.

The Scoring Metric

The main goal of a benchmark is to provide a non-biased, fair measure of how a system performs for a given task. In order to provide such a measure, one or more scoring functions are defined. These scoring functions typically measure things such as throughput or latency, to give a sense of the performance of the SUTs. However, some benchmarks define scoring functions that measure a trade-off between performance and cost, where cost is defined as the cost of operating the SUT, including the cost of purchasing the system and sometimes also its maintenance during a period of time. As a consequence, the user can know which is the best system given his budget or how much he needs to spend given the desired throughput and/or latency. Also, some use cases that allow systems to return partial or incomplete results. Thus, some benchmarks define scoring functions to measure how complete the results of a query are.

Finally, in those cases where a benchmark defines several scoring functions, it is also a common practice to define a combined metric

that unifies all such scores into a single score value in order to ease the comparison of systems.

The Auditing Rules

The third main component of a benchmark is the auditing rules. The auditing rules are a set of rules that define the conditions under which the benchmark execution must be performed in order to consider the score as valid and to ensure that it has not been gamed. The rules define various aspects such as the process to follow in order to execute the benchmark and publish its results, whether this must be executed by an independent auditor or not, what information of the SUT must be disclosed to the audience, or how to compute the cost of the SUT.

G

Key Research Findings

LUBM

The LUBM (Lehigh University Benchmark) (Guo et al. 2005) is a macro-benchmark designed for Semantic Web that has also been used to benchmark graph database systems due to the more expressive property graph data model.

The benchmark contains 14 queries crafted to be both realistic and diverse. In order to assess this diversity, the authors define a set of features for each query (e.g., *input size*, the *selectivity*, the *complexity*, etc.) and guide the definition of the queries trying to maximize the space of features covered.

The query mix consists of ten executions of each query. The benchmark uses several scoring functions, including the loading time, the repository size (i.e., the size used to store the loaded data), and the query response time per query, which are computed as the average execution time of the ten executions of each query. Additionally, the benchmark uses the notion of *completeness* and *soundness* for each query, which captures how precise and complete are the returned results (partial results are allowed). Finally, the benchmark proposes a *combined metric* that unifies all the aforementioned scoring functions into a single number.

LUBM uses a synthetic data generator, namely, the Univ-Bench Artificial (UBA) data generator, that generates datasets representing universities, their departments, the staff, the students, and the courses. The minimum unit of data generation (used to scale up the dataset) is the university, and the amount of different departments, professors, students, and courses per university is based on the authors’ “reasonable assumptions” but not backed on real data. Finally, the LUBM benchmark does not define a set of auditing rules nor any cost-based scoring function.

BSBM

The Berlin SPARQL Benchmark (Bizer and Schultz 2009) is a macro-benchmark settled in an e-commerce use case, where there are products offered by different vendors and reviews posted by consumers on various review sites. The benchmark initially targeted RDF stores, both those that execute SPARQL directly and those that rely on SPARQL-to-SQL rewritings, but it has also been used to test graph databases.

The benchmark uses a synthetic data generator to generate the different classes like product, vendor, persons, or review, just to cite a few of them. The generator scales based on the number of generated products, and for each of them, a description and reviews are generated. Then, the number of persons depends on the number of reviews generated until each review is assigned to one and only one person. The data distributions are tuned to follow the author’s “realistic assumptions,” including normal distributions for the number of reviews per product and the number of reviews per person.

The workload consists of the 12 queries that represent the interactions of users with the e-commerce site. The authors define a query mix as the succession of queries that emulates the search and navigation pattern of a consumer looking for a product, which consists of 25 query executions. The final workload is composed of different query mixes representing different users using the e-commerce site concurrently.

The BSBM benchmark uses three scoring functions: query mixes per hour (QMph) (i.e.,

the number of users served per hour), queries per second (QpS), and load time (LT), but neither a combined metric nor a cost-based metric is provided. Finally, the benchmark defines the set of steps to be properly executed, including the process of loading, system warm-up, and execution, but a formal auditing process is not specified.

LinkBench

The LinkBench (Armstrong et al. 2013) is a macro-benchmark created by Facebook to test the performance of alternative database systems when dealing with Facebook’s social graph. The benchmark replicates Facebook’s data model and the transactional query mix used on their production MySQL store.

The benchmark provides a data generator that generates a social graph similar to that of Facebook, based on statistics obtained from the original Facebook data, and that can be scaled based on the desired number of users of the social network. Similarly, the authors profiled the operations received in the MySQL back end and identified ten different types of basic operations (including read and write operations) that operate on different nodes and edges of the social graph, as well as their corresponding frequencies that conform the query mix. The workload also tries to reproduce the observed proportions of hot data (data accessed frequently) and cold data (data barely accessed).

The benchmark defines several performance metrics that need to be reported. These include the latency at 50th, 75th, 95th, and 99th percentiles and the throughput. They define the throughput per dollar as a cost/performance metric. Additionally, they suggest the gathering of resource utilization statistics (e.g., CPU usage, I/O operations, etc.) at regular intervals for understanding the performance and efficiency, but they do not provide a combined nor a cost-based scoring function. Finally, they do not define auditing rules.

LDBC SNB Interactive

The Social Network Benchmark: Interactive workload (SNB-I) (Erling et al. 2015) is a

macro-benchmark created by the Linked Data Benchmark Council, an industry and academia initiative to build benchmarks for graph-based technologies.

SNB-I models the operation of a social network site, similar to that of Facebook in terms of its data model. The SNB-I provides a synthetic data generator that simulates a social network consisting of persons, posts, comments, images, likes, groups, etc. The data generator models the characteristics typically observed in real networks, including a Facebook-like degree distribution (Ugander et al. 2011), spiky post/comment distribution based on real events (Leskovec et al. 2009), correlated attributes of entities and among connected entities (McPherson et al. 2001), etc. The benchmark defines several scale factors (SF1, SF3, …SF1000) named after the number of GB on disk that takes the generated dataset files.

The workload is a transactional workload targeting graph databases, RDBMS with graph extensions support, and RDF systems. The workload consists of 14 complex reads, which are complex yet interactive queries that explore the neighborhood of a given node; 7 short reads, which represent those short queries used to retrieve the user information like its friendships, messages, likes, etc.; and 8 update queries, which modify the state of the social network.

Complex reads are designed using a novel “choke-point”-based approach, which consists of identifying scenarios that are challenging for SUTs and designs the queries to stress such scenarios deliberately yet being use case based.

The query mix is generated at the beginning of the execution by the provided driver, such that for each query instance, a scheduled issue time is given. The query mix is designed not to make any complex read to dominate the execution, given that queries have different complexities whose runtime depends on the expected amount of data touched. Additionally, updates and their scheduled issue times are read from data provided by the data generator, while short reads are interleaved between complex reads and updates in such a way that a final read-write proportion of 80/20 is obtained.

Regarding the scoring function, the benchmark uses throughput and throughput per cost as the scoring function. Additionally, the benchmark defines a “valid run” condition, which consists of 95% of the queries to start no later than 2 s after the scheduled issue time.

Finally, the benchmark defines a comprehensive set of auditing rules that define both the process to follow to execute the benchmark (done by an official benchmark auditor) and the information that must be disclosed from the SUTs.

G

LDBC Graphalytics

The LDBC Graphalytics benchmark (Iosup et al. 2016) is an industry-backed macro-benchmark that targets graph processing systems specialized on analytics on large graphs, including distributed systems and heterogeneous systems based on GPUs.

The workload consists of six traditional structure-based (no properties are used) graph analytic algorithms: breadth-first search, PageRank, weakly connected components, community detection with label propagation, local clustering coefficient, and single-source shortest paths.

The Graphalytics benchmark uses both real graphs and synthetic graphs of different scales named after the traditional T-shirt size labeling (S, M, L, XL, XXL, etc.) that includes a renewal process that is revisited every 2 years. For generating synthetic graphs, the benchmark uses both the SNB-I data generator and the RMAT data generator.

The benchmark uses several performance metrics, including the loading time; the make-span, which is the time of executing the algorithm including the system-specific overheads; and the processing time, which is the actual time of executing the algorithm. Additionally, they collect the edges processed per second (EPS) and the edge-vertex processed per second (EVPS). The benchmark also specifies a EVPS/cost metric.

Graphalytics comprehensively defines the steps to follow to execute the full benchmark, including the experiments to run and measurements to take, as well as all the information that needs to be disclosed. So far no auditing rules

have been proposed, but their definition is on the benchmark’s development road map.

gMark

gMark (Bagan et al. 2017; gMark 2016) is a synthetic graph instance and query workload generator, which is both domain- and query language-independent. It allows the creation of domain-specific use cases by means of a highly configurable number of input parameters. Contrarily to other synthetic benchmarks, the benchmark is neither a macro-benchmark nor a micro-benchmark since it allows to simulate both cases and all the variants in between. The gMark benchmark is also workload centric as it leverages the presence of multiple queries in the workload as opposed to single-query executions. It is system- and language-independent in that it allows to generate queries for a broad variety of systems and query languages (among which SPARQL, openCypher, LogicQL, and SQL are included in the default package). It is extensible to upcoming systems and graph query languages.

The graph configuration consists of the *size* of the graph (number of nodes), the *edge predicates*, the *node types*, and the occurrence constraints (expressed as percentages of occurrence in the generated graph instance), along with the in-degree and out-degree distributions associated with each extreme of an edge. Such distributions vary among uniform, Gaussian, and Zipfian and allow to represent many realistic graph datasets. The query workload configuration allows to specify the *workload size* (in terms of number of queries), the minimum and maximum *arity* of the generated queries, the *shape* classes that are desired in the generation (out of chain, star, cycle, and star-chain queries), the *selectivity* of the queries (representing the number of results returned by the queries, which depends on the graph size and topology and can be chosen among constant, linear, and quadratic), the *probability of multiplicity* (in terms of proportion of recursive queries in the generated query workload), and the *query size* (the maximum and minimum number of conjuncts, disjuncts, etc.).

The parameters in the query workload indirectly suggest the scoring metrics to adopt in order to compare the systems under test. For instance, one can decide to measure the coverage of recursive queries of the SUTs or their robustness with respect to quadratic selectivities or to the number of queries in the workload. Nevertheless, gMark does not impose a fixed set of scoring metrics to be adopted for all cross-comparisons among the SUTs.

Finally, the gMark benchmark does not define a set of auditing rules to measure the cost of the systems under test. However, since it has been proved effective to reproduce existing benchmarks and provide expanded query workloads, such as WatDiv (Aluç et al. 2014) and LDBC SNB (Erling et al. 2015), one might adopt their respective auditing rules on the expanded query workloads.

Usage-Driven Query Analysis and Benchmarking

The rise of SPARQL endpoints has allowed to collect query logs containing manually written queries as formulated by end users and machine-generated queries through specific APIs. DBpedia 2010 logs have been examined in Morsey et al. (2011) in order to come up with a set of 25 most frequently used queries that can be considered as prototypical queries in performance studies of triple stores of the so-called DBpedia SPARQL benchmark. The identification of representative user queries has recently been carried out for Wikidata datasets and has led to define a collection of 309 Wikidata user example queries derived from real Wikidata queries (Wikidata 2018). These queries are intended to showcase systems behavior and/or highlight characteristics of the Wikidata dataset and are thus meant to produce a nice output. Recent large SPARQL query logs have been lately analyzed in Bonifati et al. (2017), encompassing DBpedia, BioPortal, SWDF, British Museum, LinkedGeoData, and OpenBioMed queries. These logs are inspected in order to understand the structural characteristics of the queries, their syntactic features, their disparate shapes, and their graph and hypergraph classification in order to understand their com-

plexity. Moreover, such true query logs contain the “development process” of queries: users start by formulating a query and gradually refining it until they are satisfied with it. Query logs can thus reveal streaks of the same evolving query, which correspond to gradual modifications of a seed query.

The above works show that much work is left to be done in order to harvest the characteristics of real user queries and feed future benchmarking initiatives devoted to target queries that users are coveting. They also highlight the importance of benchmarks that are self-adaptive to evolving users’ needs.

Future Directions of Research

Despite the fact that there has been a considerable effort from both industry and academia to create comprehensive and equitable graph benchmarks, their adoption is still far from being achieved. The graph processing industry is not yet mature for this adoption, and the number of involved industrial actors in the definition of these benchmarks is still narrow. Even though this number is increasing lately, a paradigm shift is needed in order for the involved companies to fiercely compete for the best benchmark scores. The opposite occurs in the academia, where graph processing is a hot topic and many benchmarks have been proposed as discussed in this chapter. Implementing an industry-graded benchmark is usually a time-consuming task that requires a considerable effort, not only in implementing the necessary queries but also in setting up the execution environment properly. Thus, the evolution of graph benchmarking must put a special emphasis in lowering the entry bar as much as possible, either by providing the required tools and software to ease their adoption (like workload drivers, reference query implementations, dataset repositories, etc.). A non-negligible issue is, for instance, tied to the comparison of results and their reproducibility in a benchmarking environment. Another matter concerns the definition of a standard graph query language that would

certainly help the design of valid and widely recognized benchmarks.

Another aspect is the generation of data for benchmarking purposes. The performance of graph queries is heavily affected by the characteristics of the underlying data, both structurally and in terms of attributes, which influence the size of the search space and the memory access patterns, among others. Properly understanding the relationship between different data characteristics and the performance of graph queries is one of the goals of benchmarking, and data generators that help tuning the characteristics of such data are of high importance. Moreover, the creation of tools to assist in the exploration and identification of the performance bottlenecks of graph databases and their relationship with the input data is a research direction worth to be investigated in the near future.

G

Cross-References

- ▶ [Graph Data Integration and Exchange](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Path Navigation](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)

References

- Aluç G, Hartig O, Özsu MT, Daudjee K (2014) Diversified stress testing of RDF data management systems. In: The semantic web – ISWC 2014 – Proceedings of the 13th international semantic web conference, Part I, Riva del Garda, 19–23 Oct 2014, pp 197–212
- Armstrong TG, Ponnekanti V, Borthakur D, Callaghan M (2013) Linkbench: a database benchmark based on the Facebook social graph. In: SIGMOD. ACM, pp 1185–1196
- Backstrom L, Boldi P, Rosa M, Ugander J, Vigna S (2012) Four degrees of separation. In: WebSci. ACM, pp 33–42
- Bagan G, Bonifati A, Ciucanu R, Fletcher GH, Lemay A, Advokaat N (2017) gMark: schema-driven generation of graphs and queries. IEEE TKDE 29(4):856–869
- Bizer C, Schultz A (2009) The berlin sparql benchmark. Int J Semant Web Inf Syst 5(2):1–24
- Boccaletti S, Latora V, Moreno Y, Chavez M, Hwang DU (2006) Complex networks: structure and dynamics. Phys Rep 424(4):175–308

- Bonifati A, Martens W, Timm T (2017) An analytical study of large SPARQL query logs. *VLDB* 11(2):149–161
- Chakrabarti D, Zhan Y, Faloutsos C (2004) R-mat: a recursive model for graph mining. In: SDM. SIAM, pp 442–446
- Erling O, Averbuch A, Larriba-Pey J, Chafi H, Gubichev A, Prat A, Pham MD, Boncz P (2015) The LDBC social network benchmark: interactive workload. In: SIGMOD. ACM, pp 619–630
- Girvan M, Newman ME (2002) Community structure in social and biological networks. *Proc Natl Acad Sci* 99(12):7821–7826
- gMark (2016) The gMark benchmark. <https://github.com/graphMark/gmark>
- Guo Y, Pan Z, Heflin J (2005) LUBM: a benchmark for owl knowledge base systems. *Web Semant Sci Serv Agents World Wide Web* 3(2): 158–182
- Iosup A, Hegeman T, Ngai WL, Heldens S, Prat Pérez A, Manhardt T, Chafio H, Capotă M, Sundaram N, Anderson M et al (2016) LDBC graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. *VLDB* 9(13): 1317–1328
- Kolda TG, Pinar A, Plantenga T, Seshadhri C (2014) A scalable generative graph model with community structure. *SISC* 36(5):C424–C452
- Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. *Phys Rev E* 78(4):046110
- Leskovec J, Chakrabarti D, Kleinberg J, Faloutsos C (2005) Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication. In: PKDD. Springer, Berlin/Heidelberg, vol 5, pp 133–145
- Leskovec J, Backstrom L, Kleinberg J (2009) Memetracking and the dynamics of the news cycle. In: SIGKDD. ACM, pp 497–506
- McPherson M, Smith-Lovin L, Cook JM (2001) Birds of a feather: homophily in social networks. *Ann Rev Sociol* 27(1):415–444
- Morsey M, Lehmann J, Auer S, Ngomo ACN (2011) Dbpedia sparql benchmark—performance assessment with real queries on real data. In: ISWC. Springer, pp 454–469
- Newman ME (2003) The structure and function of complex networks. *SIAM Rev* 45(2):167–256
- Newman ME, Strogatz SH, Watts DJ (2001) Random graphs with arbitrary degree distributions and their applications. *Phys Rev E* 64(2):026118
- TPC (2017) Transaction processing council. <http://www.tpc.org>
- Ugander J, Karrer B, Backstrom L, Marlow C (2011) The anatomy of the Facebook social graph. arXiv preprint arXiv:11114503
- Wikidata (2018) Wikidata SPARQL queries. http://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

Graph Grammars

► Grammar-Based Compression

Graph Invariants

Paolo Boldi

Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Synonyms

Graph centralities; Graph properties

Definitions

In this section, we cover some of the basic general definitions related to graphs, with a special emphasis on the most elementary properties of interest for link analysis; for more details and further definitions and results, see, for example, (Bollobás 1998; Diestel 2012).

Overview

Graphs are one of the basic representation tools in many fields of pure and applied science, useful to model a variety of different situations and phenomena. Besides the ubiquity of graphs as a representation tool, the emergence of social networking and social media has given a new strong impetus to the field of “social network analysis” and, more generally, to what is called “link analysis.” In a nutshell, link analysis aims at studying the graph structure in order to unveil properties, discover relations, and highlight pattern and trends. This process can be seen as a special type of data mining (Chakrabarti et al. 2006), sometimes referred to as graph mining.

A (*directed*) graph (or digraph, or network) G consists of a finite set of nodes N_G and a set

of *arcs* $A_G \subseteq N_G \times N_G$ (the subscript, here and in the following, is omitted whenever it can be deduced from the context). According to this definition, a graph is given by a certain ensemble of objects (its “nodes,” whose nature depends on the actual context) and a binary relation between them, represented by its arcs. An arc of the form $\langle x, x \rangle$ is called a *(self-)loop*; unless otherwise specified, loops are allowed to exist.

The adjective “directed” emphasizes that arcs have a direction (they go *from* one node *to* another); in some applications, though, the direction does not make sense, and one talks of “undirected graphs,” i.e., graphs whose arcs do not possess an orientation. In the context of undirected graphs, nodes are often called *vertices*, and arcs are called *edges*. An *undirected graph* can be alternatively defined as a loopless graph such that if $\langle x, y \rangle$ is an arc then also $\langle y, x \rangle$ is one (the pair of opposite arcs represents a single edge).

A few examples of directed and undirected networks that arise in different scientific contexts include the following: hypertextual document collections (the most famous instance being the World Wide Web), computer communication networks (among them, the network of Internet autonomous systems), telephone call networks, email-exchange networks, social interaction networks (of different kinds and over a variety of social networking applications and websites), biological networks (e.g., protein-protein interaction networks, metabolic networks, etc.), road-connection networks...

We let $n = |N|$ and $m = |A|$ be the number of nodes and arcs of the graph, respectively; for undirected graphs, we write $e = |A|/2$ for the number of edges. We also let \mathcal{G} denote the family of all graphs.

Note that $0 \leq m \leq n^2$ for directed graphs (and $0 \leq e \leq \binom{n}{2}$ for undirected graphs): if $m = 0$, we say that the graph is an *independent set*; if $m = n^2$ (or $e = \binom{n}{2}$, in the undirected case), we say it is a *clique* (or complete graph).

An *out-neighbor* (*in-neighbor*, respectively) of the node x is any node y such that $\langle x, y \rangle$ ($\langle y, x \rangle$, respectively) is an arc. The set of out-neighbors (in-neighbors, respectively) of x is

denoted by $N^+(x)$ ($N^-(x)$, respectively). For undirected graphs, we just talk of *neighbors* of x and write $N(x)$.

The number of out-neighbors (in-neighbors, neighbors) of x is called its *out-degree* (*in-degree*, *degree*), and it is denoted by $d^+(x)$ ($d^-(x)$, $d(x)$).

Given a set of nodes $X \subseteq N$, the *subgraph induced by X* is the graph $G[X]$ with node set X and with arcs given by $A \cap (X \times X)$.

A *k -partite graph* is a graph whose nodes can be partitioned into k disjoint subsets N_1, \dots, N_k such that, for all $i = 1, \dots, k$, $G[N_i]$ is an independent set.

G

Invariants and Centrality Indices

In some cases, the actual nature of the nodes of a graph is important and relevant, for example, when drawing an undirected graph, its vertices are (identified with) points in the Euclidean plain, and its edges are segments (or curves) connecting them; moving one vertex to another location would change the drawing of the graph.

Nonetheless, in many cases one is not interested in the concrete attributes of the graph nodes, but rather in the graph “structural” (or “topological”) properties, the ones that can be deduced solely from the organization of its arcs. Formally, those properties are called invariants; to define them, we first need to say what a graph morphism is.

Given two graphs $G = (N_G, A_G)$ and $H = (N_H, A_H)$, a *graph morphism* is a function $f : N_G \rightarrow N_H$ such that $\langle x, y \rangle \in A_G$ iff $\langle f(x), f(y) \rangle \in A_H$. If f is a bijection, we say that f is a graph *isomorphism*; if such an isomorphism exists, we say that G and H are isomorphic and write $G \cong H$.

A *V -valued graph invariant* (Lovász 2012) is any function $\pi : \mathcal{G} \rightarrow V$ such that $G \cong H$ implies $\pi(G) = \pi(H)$: a graph invariant is a property of a graph that does not depend on its actual representation (i.e., on the names of its nodes) but only on the graph structure. Binary invariants are those for which $V = \{\text{true}, \text{false}\}$: for

instance, properties like “the graph is connected,” “the graph is planar,” and so on are all examples of binary graph invariants. Scalar invariants have values on a field (e.g., $V = \mathbb{R}$), for instance, “the number of connected components,” “the average length of a shortest path,” and “the maximum size of a connected component,” are all examples of scalar graph invariants. Distribution invariants take as value a *distribution*, for instance, “the degree distribution” or the “shortest-path-length distribution” are all examples of distribution invariants.

It is convenient to extend the definition of graph invariants to the case where the output is in fact a function assigning a value to each node. Formally, a V -valued *node-level graph invariant* is a function π that maps every $G \in \mathcal{G}$ to an element of V^{N_G} (i.e., to a function from N_G to V), such that for every isomorphism $f : G \rightarrow H$ one has $\pi(G)(x) = \pi(H)(f(x))$.

For example, suppose that π assigns to every graph G the function that maps each node $x \in N_G$ to the number of triangles (closed cycles of length three) involving x ; if $G \cong H$ and $x' \in N_H$ is the node of H that corresponds to $x \in N_G$, the number of triangles of H involving x' is the same as the number of triangles of G involving x , that is, $\pi(G)(x) = \pi(H)(x')$.

Introducing node-level graph invariants is crucial in order to be able to talk about graph centralities (Newman 2010). A node-level real-valued graph invariant that aims at estimating node importance is called a *centrality (index or measure)*; given a centrality index $c : N_G \rightarrow \mathbb{R}$, we interpret $c(x) > c(y)$ as a sign of the fact that x is “more important” (more central) than y . Since the notion of being “important” can be declined to have different meanings, in different contexts, many indices were proposed in the literature (Boldi and Vigna 2014); some of the most important ones will be mentioned in this section.

A notable, simple example of centrality in an undirected graph is simply node degree; in a friendship network, degree centrality amounts at measuring the importance of a node (person) by the number of friends (s)he has. As simple as it may seem, degree contains a lot of information,

and in some contexts it is actually almost as good as other more sophisticated alternatives (Craswell et al. 2003). We take this opportunity to notice that most centrality measures proposed in the literature were actually described originally only for undirected graphs; adapting those measures to the directed case often implies to choose between looking at incoming or outgoing paths: in the case of degree, for example, we should choose whether to consider in-degrees or out-degrees. In most cases, the so-called “negative” version (the one that considers incoming paths) is more meaningful than the “positive” alternative (Boldi and Vigna 2014). In-degree, for instance, is probably the oldest measure of importance ever used, as it is equivalent to majority voting in elections (where the presence of an arc $\langle x, y \rangle$ means that x voted for y).

Local Properties

The *average degree* of a graph m/n and its *density* m/n^2 ($e/\binom{n}{2}$) in the case of undirected graphs) determine how close the graph is to being a clique. A family of graphs is said to be *sparse* if their average degree is $o(n)$, *dense* if it is $\Theta(n)$.

A more fine-grained invariant measuring the density of a graph is given by its full *(in/out-)degree distribution*, a discrete distribution whose probability mass function evaluated at k is the fraction of nodes with (in/out-)degree k .

The degree distribution is quite distinctive and contains by itself much information about the structure of the graph. The degrees of a random graph (in the sense of Erdős and Rényi 1959) follow a binomial distribution. It was noted, however, that many networks corresponding to empirical phenomena have a peculiarly different behavior and were called scale-free: a graph is *scale-free* (Barabási and Albert 1999) if its degree distribution follows a power law, that is, if the cumulative function $F(\cdot)$ of its degree distribution has the property that

$$1 - F(x) \propto x^{-\gamma}$$

for some constant $\gamma > 1$.

The degree of a node tells us how many direct connections a node has but does not give any information about how much those neighbors are connected with one another. An informative index in this direction (a node-level graph invariant itself) is the *clustering coefficient* (Watts and Strogatz 1998) of node x , defined for undirected graphs by

$$\frac{\sum_{y \in N(x)} |N(y) \cap N(x)|}{d(x)(d(x) - 1)}.$$

This is the probability that two neighbors of x chosen at random turn out to be connected by an edge: its extreme values are 0 if $G[N(x)]$ is an independent set and 1 if $G[N(x)]$ is a clique. It is possible to extend this definition to the directed case, but we shall not discuss this extension here.

Of course the *distribution of clustering coefficients* can be used as a graph invariant; in particular, its average is a quite distinctive network feature, called *global clustering coefficient* (Newman 2003). An alternative (albeit not quite equal) definition used by some authors is the ratio between the number of closed triplets (i.e., induced complete subgraphs with three nodes) and the number of connected triplets (i.e., induced subgraphs with three nodes none of which has degree zero in the subgraph).

Paths and Connectivity

A *path* of length $k \geq 0$ from node x to node y is a sequence of nodes v_0, v_1, \dots, v_k such that $x = v_0$, $y = v_k$ and $\langle v_{i-1}, v_i \rangle \in A$ for all $i = 1, \dots, k$. A path is called a *cycle* if $k > 0$ and $x = y$ (in the undirected case it is also required that $v_{i-1} \neq v_{i+1}$ for all $i = 1, \dots, k - 1$). A graph with no cycles is called *acyclic*; in particular, if it is directed, it is called a *DAG* (directed acyclic graph); if it is undirected, it is called a *forest*.

A graph is *connected* if there is a path between every ordered pair of nodes; note that, in the case of directed graphs, this requirement implies that for any two nodes x and y , there is both a path from x to y and a path from y to x . Maximal sets of nodes that induce a connected subgraph

are pairwise disjoint, so they define a partition of the node set whose parts are called *connected components*.

In the case of undirected graphs, there can exist no edge between vertices belonging to different components; this is not true for directed graphs, but a similar property is true of cycles (there can be no cycles involving nodes in different components). An undirected connected acyclic graph (i.e., a connected forest) is called a *tree*.

The number of paths ending at a node x can be used to estimate its importance in the network; for example, *Katz's index* (Katz 1953) of node x is defined as

$$\sum_{t=0}^k \beta^t |G^t(-, x)|$$

where $G^t(-, x)$ is the set of paths of length t ending in x and β is a parameter that must be chosen so that $\beta < 1/\rho$ where ρ is the spectral radius of the adjacency matrix G (Gross and Tucker 1987).

Shortest Paths

A *shortest (or geodesic) path* is a path of shortest length connecting two given nodes; the *distance* $d(x, y)$ from node x to node y is the length of (any) shortest path from x to y or $+\infty$ if there is no such path. Note that $d(x, y) = d(y, x)$ if the graph is undirected, whereas this is not true in general for directed graphs.

For every node x , the maximum distance from x to any other node $\max_y d(x, y)$ is called the *out-eccentricity* of x ; in-eccentricity is defined analogously; in the undirected case they of course coincide and are simply called "eccentricity." The maximum eccentricity (i.e., the maximum distance between any two nodes) is called the *diameter* of the graph; the diameter is finite only if the graph is strongly connected. Eccentricity (i.e., maximum distance to/from a node), or more precisely its reciprocal, is a natural centrality index, but it is very sensitive to outliers;

a smoother alternative is closeness centrality that corresponds, up to constants, to (the reciprocal of the) average distance. More precisely, the *closeness centrality* (Bavelas 1948) of x is defined as

$$\frac{1}{\sum_y d(y, x)}.$$

The summation ranges over the nodes y for which $d(y, x)$ is finite (otherwise, in non-strongly connected graphs, almost all nodes would have null centrality). An alternative definition, which works better in the non-strongly connected case, is *harmonic centrality* (Harris 1954; Boldi and Vigna 2014), defined for a node x as

$$\sum_y \frac{1}{d(y, x)}$$

where this time the summation is over all $y \neq x$, with the proviso that $1/\infty = 0$.

Eccentricity, closeness, and harmonic may all be seen as variants of the same idea of measuring centrality the centrality of a node x through the set of distances $d(-, x)$: such centralities are called *geometric* (Boldi and Vigna 2014). A quite popular centrality index, which is also related to shortest paths, but not to distances in the strict sense, is *betweenness* (Anthonisse 1971; Freeman 1977); the betweenness centrality of a node x is defined as

$$\sum_{u,v} \frac{\sigma_{u,v}(x)}{\sigma_{u,v}}$$

where $\sigma_{u,v}$ is the number of shortest paths from u to v and $\sigma_{u,v}(x)$ is the number of such paths that pass through x ; the summation is extended over all pairs u, v (both distinct from x) such that $u \neq v$ and there is a path from u to v .

Special Invariants

Spectral graph theory (Cvetkovic and Rowlinson 2004) studies invariants that are related to the

eigenvalues and eigenvectors of the adjacency matrix of the graph or of some matrices derived from it. There is by now an immense body of literature showing the deep and unexpected connections between spectral properties and topological structure, in some cases leading to interesting node-level invariants.

The first and most obvious spectral centrality (called the *Wei-Berge index*) is the left dominant eigenvector of the adjacency matrix G itself (Berge 1958). But other centrality measures are of spectral nature; for instance, it turns out that Katz's index is the left dominant eigenvector of a perturbed version of the adjacency matrix:

$$\beta\lambda G + (1 - \beta\lambda)\mathbf{e}^T \mathbf{1}, \quad (1)$$

where λ is the dominant eigenvalue of G and \mathbf{e} is a right dominant eigenvector of G such that $\mathbf{1}\mathbf{e}^T = \lambda$ (Vigna 2016).

As another example, define \bar{G} as the matrix obtained by dividing each nonnull row of G by its L^1 -norm and substituting null rows with $\mathbf{1}/n$. The left dominant eigenvector of the matrix

$$\alpha\bar{G} + (1 - \alpha)(\mathbf{1}/n)^T \mathbf{1}, \quad (2)$$

is the well-known *PageRank* centrality index (Brin and Page 1998). Since \bar{G} has dominant eigenvalue 1, the analogy with (1) should be clear. The popularity of PageRank is related to the fact that it was supposedly used by Google to rank websites in their search engine results.

Alternative Definitions

The graph definitions considered in this section (directed and undirected graphs) are only the two most important representatives of a wide spectrum of objects that are subtly different and may be called themselves “graphs” or “networks” without further specification, largely depending on the research area and context.

A *weighted graph* is a graph endowed with a weighting function that assigns a weight (a

positive real number, usually) to each node or arc (edge, in the case of undirected graphs). Weighted graphs appear in a variety of situations, and typically one wants to take weights into account when dealing with a weighted graph. All the definitions and properties discussed above can be adapted to the weighted case. For example, the length of a path in an arc-weighted graph is usually defined as the sum of the weights of the arcs along the path; obviously, under this definition, shortest paths have no longer the minimum number of arcs, and the distance between two nodes is not an integer anymore, in general. In some cases, the modification needed to deal with the weighted case is straightforward; in other cases, some domain knowledge (of the meaning behind weights) is required to decide how the definitions should be modified.

An *uncertain graph* (Jin et al. 2011) is special type of weighted graph whose weights are interpreted as probabilities (and, therefore, are constrained to be values between 0 and 1). An uncertain graph U with n nodes may be interpreted as a compact way to define a distribution over the graphs of n nodes, in which only arcs of U may appear and every arc is drawn independently with a probability equal to its weight. They are a natural generalization of Erdős-Rényi random graphs (Erdős and Rényi 1959) (that are equivalent to uncertain cliques with constant weights).

A *multigraph* (Gross and Tucker 1987) is a graph whose arcs are a multiset and not a set; in other words, there may be more than one arc connecting two nodes. Again, many of the definitions carry over to this case without almost any effort.

A k -*hypergraph* (Berge 1984) is an undirected graph whose edges (called “hyperedges”) are not pairs, but rather sets of nodes of cardinality k .

References

- Anthonisse JM (1971) The rush in a directed graph. Technical Report BN 9/71, Mathematical Centre, Amsterdam

- Barabási AL, Albert R (1999) Emergence of scaling in random networks. *Science* 286(5439):509–512
- Bavelas A (1948) A mathematical model for group structures. *Hum Organ* 7:16–30
- Berge C (1958) Théorie des graphes et ses applications. Dunod, Paris
- Berge C (1984) Hypergraphs: combinatorics of finite sets, vol 45. Elsevier, North Holland
- Boldi P, Vigna S (2014) Axioms for centrality. *Internet Math* 10(3–4):222–262
- Bollobás B (1998) Modern graph theory. Graduate texts in mathematics. Springer, Heidelberg
- Brin S, Page L (1998) The anatomy of a large-scale hypertextual Web search engine. *Comput Netw ISDN Syst* 30(1):107–117
- Chakrabarti S, Ester M, Fayyad U, Gehrke J, Han J, Morishita S, Piatetsky-Shapiro G, Wang W (2006) Data mining curriculum: a proposal (version 1.0). In: Intensive working group of ACM SIGKDD curriculum committee, p 140
- Craswell N, Hawking D, Upstill T (2003) Predicting fame and fortune: PageRank or indegree? In: In Proceedings of the Australasian document computing symposium, ADCS2003, pp 31–40
- Cvetkovic D, Rowlinson P (2004) Spectral graph theory. In: Beineke LW, Wilson RJ, Cameron PJ (eds) Topics in algebraic graph theory, vol 102. Cambridge University Press, Cambridge, p 88
- Diestel R (2012) Graph theory, 4th edn. Graduate texts in mathematics, vol 173. Springer, New York
- Erdős P, Rényi A (1959) On random graphs, I. *Publicationes Mathematicae (Debrecen)* 6:290–297
- Freeman LC (1977) A set of measures of centrality based on betweenness. *Sociometry* 40(1):35–41
- Gross JL, Tucker TW (1987) Topological graph theory. Series in discrete mathematics and optimization. Wiley, New York
- Harris CD (1954) The market as a factor in the localization of industry in the United States. *Ann Assoc Am Geogr* 44(4):315–348
- Jin R, Liu L, Aggarwal CC (2011) Discovering highly reliable subgraphs in uncertain graphs. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 992–1000
- Katz L (1953) A new status index derived from sociometric analysis. *Psychometrika* 18(1):39–43
- Lovász L (2012) Large networks and graph limits. Colloquium publications, vol 60. American Mathematical Society, Providence
- Newman M (2003) The structure and function of complex networks. *SIAM Rev* 45:167–256
- Newman MEJ (2010) Networks: an introduction. Oxford University Press, Oxford/New York
- Vigna S (2016) Spectral ranking. *Netw Sci* 4(4):433–445
- Watts D, Strogatz S (1998) Collective dynamics of ‘small-world’ networks. *Nature* 393(6684):440–442

Graph Layout

► Graph Visualization

Graph OLAP

Alejandro A. Vaisman¹ and Esteban Zimányi²

¹Instituto Tecnológico de Buenos Aires (ITBA),
Buenos Aires, Argentina

²CoDE, Université Libre de Bruxelles, Brussels,
Belgium

Synonyms

Graph aggregation; Graph analytics; Graph Summarization

Definitions

Online analytical processing (OLAP) comprises tools and algorithms that allow querying multidimensional databases, where data can be seen as a cube and each cell contains one or more measures that can be aggregated along dimensions. A graph database management system (GDMS) allows creating, reading, updating, and deleting a graph data model. Graph databases provide better support for graph data management than relational databases, mainly due to the fact that relational databases deal just implicitly with connected data, while graph databases store actual graphs. Applying OLAP techniques to graph databases is denoted Graph OLAP.

Overview

Two graph database models are used in practice:

- (a) Models based on RDF (Hayes and Patel-Schneider 2014), oriented to the Semantic

Web. A practical example of this model is AllegroGraph.

- (b) The models based on Property Graphs. An example of this model is Neo4J.

Models of type (a) represent data as sets of triples, where each triple consists of three elements that are referred to as the subject, the predicate, and the object of the triple. These triples allow describing arbitrary objects in terms of their attributes and their relationships to other objects. An important feature of RDF-based graph models is that they follow a standard, which is not the case for the other graph databases at the moment of writing this entry. In the *Property Graph* data model, nodes and edges are labeled with a collection of (attribute-value) pairs. Hartig (2014) presented a formal way of reconciling both models through a collection of well-defined transformations between Property Graphs and RDF. In that work, the author proves that Property Graphs could, in the end, be queried using SPARQL (Harris and Seaborne 2012), the standard query language for the Semantic Web.

Graph data management systems can also be classified in many other ways. For example, *graph databases* are in general adequate for medium-sized graphs, since they do not partition the graph for parallel processing. When there is the need to handle very large graphs, the so-called graph processing frameworks are used. Further, graphs whose nodes are of the same kind are denoted *homogeneous*. Opposite to this, graphs that can have nodes of different kinds are called *heterogeneous*. Finally, graphs can be classified according to their temporal support: graphs that do not change across time are called *static*, and they basically represent a snapshot of a network at a certain moment in time. However, in real-world applications, graphs change over time. A typical example is social networks, where people add and delete connections and/or change communication patterns. This extends to almost every application field. Graphs accounting for the time dimension are called *dynamic*, and they are aimed at representing the history of the network. Sometimes they are also called

temporal graphs. A particular and novel variation of dynamic graphs are *stream graphs*, where edge (and possibly node) streams arrive in real time.

Graph analytics has been steadily gaining momentum in the data management community, since many real-world applications are naturally modeled as graphs, in particular in the domain of social network analysis. Given the extensive use of graphs to represent practical problems, multidimensional analysis of graph data and graph data warehouses are promising fields for research and application development. There is a need to perform graph analysis from different perspectives and at multiple granularities. This poses new challenges to the traditional OLAP technology. This is generically called Graph OLAP. The remainder of this entry will address research on this topic.

Key Research Findings

This section starts with a review on graph summarization. Graph OLAP is then discussed. Finally, relevant contributions on analytics over dynamic graphs are presented.

Graph Summarization

Graph summarization is a critical operation for multilevel analysis of graph data and must be accounted for when addressing OLAP on graphs. However, early work on graph summarization was aimed at different goals, mainly trying to find a smaller representation of the input graph, which can reveal patterns in the original data preserving specific structural properties. This is why graph summarization is also called graph coarsening. Thus, graph summarization aims at (a) reducing the data volume producing a graph summary of the original graph, (b) speeding up algorithms, (c) noise elimination, and (d) effective visualization. Graph summarization will be discussed below, although its goal is in some sense broader than the one of Graph OLAP, which mainly refers to exploratory and interactive data analysis on graphs.

The work by Tian et al. (2008) introduces the SNAP operation (standing for summarization by grouping nodes on attributes and pairwise relationships) to produce a summary graph by grouping nodes based on node attributes and relationships selected by the user. The k-SNAP operation is proposed, allowing drilling down and rolling up at different aggregation levels. The work also presents an algorithm to evaluate the SNAP operation, shows that the k-SNAP computation is NP complete, and proposes two heuristic methods to approximate the k-SNAP results. Nodes are aggregated based on the strength of the relationships between them.

OLAP on Graph Data

A first approach to performing OLAP on graphs is called *Graph OLAP* (Chen et al. 2009) and presents a multidimensional and multilevel view of graphs. To illustrate the idea, consider a set of authors working in a given field, say data warehouses. If two persons coauthored x papers in the DaWaK 2009 conference, a link is added between them, which has a collaboration frequency attribute x . For every conference in every year, there is a coauthor graph describing the collaboration patterns between researchers. Thus, each graph can be viewed as a snapshot of the overall collaboration network. These graphs can be aggregated in an OLAP style. For instance, graphs can be aggregated to obtain the collaborations by conference type and year for all pairs of authors. For this, nodes and edges in each snapshot graph must be aggregated according to the conference type (e.g., database conferences) and by year. For example, if there is a link between two authors in the SIGMOD and VLDB conferences, the nodes and the edges will be in the aggregated graph corresponding to the conference type "Databases." More complex patterns can be obtained, for example, by merging the authors belonging to the same institution, enabling to obtain patterns of collaboration between researchers of the same institutions. In *Graph OLAP*, dimensions are classified as *informational* and *topological*. The former are close to traditional OLAP dimension hierarchies using information of the snapshot levels, for example,

Conference → Field → All. They can be used to aggregate and organize snapshots as explained above. Topological dimensions can be used for operating on nodes and edges within individual networks. For example, a hierarchy for authors like **AuthorId → Institution** will belong to a topological dimension since author institutions do not define snapshots. These definitions yield two different kinds of *Graph OLAP* operations. A roll-up over an informational dimension overlays and joins snapshots (but does not change the objects), while a roll-up over a topological dimension merges nodes in a snapshot, modifying its structure.

Graph Cube (Zhao et al. 2011) is a model for graph data warehouses that supports OLAP queries on large multidimensional networks, accounting for both, attribute aggregation and structure summarization of the networks. A multidimensional network consists of a collection of vertices each one of which contains a set of multidimensional attributes describing the node properties. For example, in a social network, nodes can represent persons, and multidimensional attributes may include **UserID**, **Gender**, **City**, etc. Thus, multidimensional attributes in the graph vertices define the dimensions of the graph cube. Measures are aggregated graphs summarized according to some criteria. Opposite to *Graph OLAP*, where the model consists in several snapshots, in *Graph Cube* there is only one large network, leading to a graph summarization problem. For example, consider a small social network with three nodes, two of them corresponding to male individuals in the network, while the other one corresponds to a female. A graph that summarizes the connections between genders will have two nodes, one labeled **male** and the other labeled **female**. The edges between them will be annotated with the number of connections of some kind. For instance, if in the original graph there were two connections between two male persons (in both directions), the summarized graph will contain a cycle over the **male** node, annotated with a “2.” If there was just one connection between a woman and a man, there will be an edge between nodes **male** and **female** annotated with a “1.”

Based on the idea of defining graphs at different levels of aggregation, denoted graph cuboids, *Distributed Graph Cube* (Denis et al. 2013) presents a distributed framework for graph cube computation and aggregation, implemented using Spark and Hadoop. This proposal addresses homogeneous graphs. Along the same lines, *Pagrol* (Wang et al. 2014) introduces a MapReduce framework for distributed OLAP analysis of homogeneous attributed graphs. *Pagrol* extends the model of *Graph Cube* by considering the attributes of the edges as dimensions. The idea behind these two proposals is similar: to efficiently compute all possible aggregations of a homogeneous graph.

To handle heterogeneous graphs, and also starting from the topological and informational dimensions introduced in Chen et al. (2009), Yin et al. (2012) proposed a data warehousing model based on the notion of entity dimensions. In this work, two basic operations are defined, namely, rotate and stretch, which allow defining different views of the same graph, defining entities as relationships and vice versa.

Gómez et al. (2017) propose a formal multidimensional data model for graph analysis, where graphs are node- and edge-labeled directed multi-hypergraphs, called *graphoids*, defined at several different levels of granularity, according to dimension hierarchies associated with them. Over this model, the authors define Graph OLAP operations. This model addresses heterogeneous graphs.

Dynamic Graph Analytics

The methods discussed above apply, basically, to static graphs. The work on dynamic graph analytics has mainly focused in graph mining rather than in graph summarization and temporal Graph OLAP. However, some works on graph summarization and temporal OLAP are briefly commented below.

An early approach to dynamic graph summarization consisted in creating an aggregate graph that summarizes the input dynamic one, based on how recent and frequent were the interactions. This is called an *approximation graph* (Hill et al. 2006; Sharan and Neville 2008). The approxi-

mation graph is built aggregating the node interactions across time, giving a weight to them, and this weight is higher when the interactions are more recent and frequent. These solutions are used for mining and visualization. Another approach is based on compression techniques. This is the case of *TimeCrunch* (Shah et al. 2015), which aims at extracting meaningful patterns from temporal data, in order to describe a dynamic graph. The method first generates static subgraph instances in each defined time stamp, labels each instance using a dictionary of static templates (e.g., star, clique, chain), and puts them together to form temporal instances. Then, it labels each structure using a dictionary of temporal signatures (e.g., one-shot, ranged, periodic). Finally, it selects for the summary the temporal structures that produces the shortest lossless decomposition (MDL) for describing the time-evolving graph.

A temporal graph model has been presented in Cattuto et al. (2013), where temporal data are organized in so-called frames, namely, the finest unit of temporal aggregation. A frame is associated with a time interval and allows to retrieve the status of the social network during such interval. A limitation of the model is that it does not allow registering changes in attributes of the nodes and that frame nodes become too cluttered with edges.

A proposal for temporal OLAP on graphs was introduced by Campos et al. (2016). Here, a temporal model for graphs is presented, based on Property Graphs, along with a query language, which extends Cypher with temporal capabilities.

Examples of Applications

Graph summarization can be applied in several different settings. On the technical side, for improving graph query efficiency, for example, some queries can be answered using a summary graph instead of the whole one. This happens in social network analysis when issuing a typical query to compute whether an edge exists between two nodes, which can be evaluated more

efficiently over a summary rather than on the whole network. Summarization is also relevant for graph visualization since, normally, graphs are too large to visualize in an effective way. Pattern discovery is also a field of application of graph summarization and Graph OLAP. Patterns or outliers that are not immediately noticeable over the whole graph can become evident on certain summarizations along some dimensions.

G

Future Directions of Research

There is a consensus in the database community in that novel models and techniques are required for enabling multidimensional analysis of big data (Cuzzocrea et al. 2013). From the discussion in this entry, it follows that several challenges arise: on the one hand, traditional data warehousing and OLAP operations on cubes are clearly not sufficient to address the current data analysis requirements; on the other hand, OLAP operations and models can expand the possibilities of graph analysis beyond the traditional graph-based computation, like shortest-path, centrality analysis and so on. Nevertheless, from the discussion in the present entry, it is clear that there is still no consensus on what Graph OLAP means. Each proposal commented above presents a different model, with different operators. Therefore, the main challenge for the OLAP and databases communities is to agree in the model and operators for Graph OLAP.

Many challenges in Graph OLAP occur in the field of dynamic graphs. As expressed above, summarization techniques for these kinds of graphs have not been studied as in the case of static graphs. The addition of the time dimension introduces many problems, not only theoretical but also practical. Analogously to the case of temporal relational databases, recording the history of graph data involves enormous amounts of data, which is also highly dependent on the time granularity that is chosen. Temporal query languages for graphs and query processing techniques must be developed. Temporal graph summarization also opens many

research opportunities in the graph visualization field. Finally, stream processing for (dynamic) graphs is still in its infancy, and there is a fertile field of research there.

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Processing Frameworks](#)
- ▶ [Graph Query Processing](#)
- ▶ [Historical Graph Management](#)
- ▶ [Visual Graph Querying](#)

References

- Campos A, Mozzino J, Vaisman AA (2016) Towards temporal graph databases. In: Proceedings of the 10th Alberto Mendelzon international workshop on foundations of data management, CEUR-WS.org, CEUR workshop proceedings, vol 1644
- Cattuto C, Quaggiotto M, Panisson A, Averbuch A (2013) Time-varying social networks in a graph database. In: Proceedings of the 1st international workshop on graph data management experiences and systems, p 11
- Chen C, Yan X, Zhu F, Han J, Yu P (2009) Graph OLAP: a multi-dimensional framework for graph data analysis. *Knowl Inf Syst* 21(1):41–63
- Cuzzocrea A, Bellatreche L, Song IY (2013) Data warehousing and OLAP over big data: current challenges and future research directions. In: Proceedings of the 16th international workshop on data warehousing and OLAP. ACM, pp 67–70
- Denis B, Ghrab A, Skhiri S (2013) A distributed approach for graph-oriented multidimensional analysis. In: Proceedings of the IEEE international conference on big data, pp 9–16
- Gómez LI, Kuijpers B, Vaisman AA (2017) Performing OLAP over graph data: query language, implementation, and a case study. In: Proceedings of the international workshop on real-time business intelligence and analytics, pp 6:1–6:8
- Harris S, Seaborne A (2012) SPARQL 1.1 query language. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>
- Hartig O (2014) Reconciliation of RDF* and property graphs. CoRR abs/1409.3288. <http://arxiv.org/abs/1409.3288>
- Hayes P, Patel-Schneider PF (2014) RDF semantics. <http://www.w3.org/TR/rdf-mt/>
- Hill S, Agarwal D, Bell R, Volinsky C (2006) Building an effective representation for dynamic networks. *J Comput Graph Stat* 15(3):1–25
- Shah N, Koutra D, Zou T, Gallagher B, Faloutsos C (2015) TimeCrunch: interpretable dynamic graph summarization. In: Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining, pp 1055–1064
- Sharan U, Neville J (2008) Temporal-relational classifiers for prediction in evolving domains. In: Proceedings of the 8th IEEE international conference on data mining, pp 540–549
- Tian Y, Hankins RA, Patel JM (2008) Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, pp 567–580
- Wang Z, Fan Q, Wang H, Tan KL, Agrawal D, Abbadi AE (2014) Pagrol: parallel graph OLAP over large-scale attributed graphs. In: Proceedings of the IEEE 30th international conference on data engineering, pp 496–507
- Yin M, Wu B, Zeng Z (2012) HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis. In: Proceedings of the 15th international workshop on data warehousing and OLAP. ACM, pp 137–144
- Zhao P, Li X, Xin D, Han J (2011) Graph cube: on warehousing and OLAP multidimensional networks. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data. ACM, pp 853–864

Graph Parallel Computation API

- ▶ [Graph Processing Frameworks](#)

Graph Partitioning: Formulations and Applications to Big Data

Christian Schulz¹ and Darren Strash²

¹University of Vienna, Vienna, Austria

²Department of Computer Science, Colgate University, Hamilton, NY, USA

Definitions

Given an input graph $G = (V, E)$ and an integer $k \geq 2$, the *graph partitioning problem* is to divide V into k disjoint blocks of vertices V_1, V_2, \dots, V_k , such that $\cup_{1 \leq i \leq k} V_i = V$, while

simultaneously optimizing an objective function and maintaining balance: $|V_i| \leq (1 + \epsilon) \lceil |V|/k \rceil$ for some $\epsilon \geq 0$.

Overview

Subdividing a problem into manageable pieces is a critical task in effectively parallelizing computation and even accelerating sequential computation. A key method that has received a lot of attention for doing so is *graph partitioning*. The simplest and most common form of graph partitioning asks for the vertex set to be partitioned into k roughly equal-sized blocks while minimizing the number of edges between the blocks (called *cut edges*). Even this most basic variant is NP-hard (Hyafil and Rivest 1973). Graph partitioning has many application areas, including VLSI (Karypis et al. 1999), scientific computing (Langguth et al. 2015), designing scalable distributed systems (Shalita et al. 2016), and genome sequencing (Jammula et al. 2017). Given its usage in many varied areas, it is critical to understand how a high-quality graph partition can be computed and used in practice.

Here, we give a structured overview of different graph partitioning formulations and applications in the *big data* context. More specifically, we give in-depth prototypical applications in this context and outline points where application-specific tailor-made graph partitioning algorithms may exist or would be highly useful. For a more detailed picture on how the field has evolved, we refer the interested reader to a number of surveys (Bichot and Siarry 2011; Schloegel et al. 2003; Buluç et al. 2016; Kim et al. 2011).

Preliminaries

In the remainder of the entry, we use the following notation. A graph $G = (V, E, c, \omega)$ consists of a vertex set $V = \{0, 1, \dots, n - 1\}$ of n vertices and a set $E \subseteq \{\{u, v\} : u, v \in V\}$ of m undirected edges formed between pairs of vertices. Nodes and edges may have a real-

valued weight, given by $c : V \rightarrow \mathbb{R}_{\geq 0}$, $\omega : E \rightarrow \mathbb{R}_{> 0}$. For brevity, c and ω are defined on sets as $c(V' \subseteq V) := \sum_{v \in V'} c(v)$ and $\omega(E' \subseteq E) := \sum_{e \in E'} \omega(e)$. For unweighted graphs, node and edge weights are one. We further let $d(v)$ be the *degree* of the vertex v , which is the size of v 's neighborhood $N(v) = \{u \in V : \{u, v\} \in E\}$.

G

Key Research Findings

The goal of graph partitioning is to partition the graph into roughly equal-sized subgraphs, called *blocks*, while optimizing an objective function. Many flavors of graph partitioning exist and can be categorized by the objectives and the graph primitives (vertices or edges) that make up the blocks.

Vertex-Based Partitioning

Vertex-based partitioning is the most common formulation of graph partitioning. Given an input graph $G = (V, E)$ and an integer $k \geq 2$, V must be divided into k disjoint blocks of vertices V_1, V_2, \dots, V_k , such that $\cup_{1 \leq i \leq k} V_i = V$ and $V_i \cap V_j = \emptyset, \forall i \neq j$, which is called a k -way partition. The goal is to minimize (or maximize) an objective function, subject to the constraint that blocks are balanced according to the *balancing constraint*: for all $i \in \{1, \dots, k\}$: $|V_i| \leq L_{\max} = (1 + \epsilon) \lceil |V|/k \rceil$ for some imbalance parameter $\epsilon \geq 0$. The most common objective is to minimize the total *cut*, $\sum_{i < j} \omega(E_{ij})$, where $E_{ij} = \{\{u, v\} \in E : u \in V_i, v \in V_j\}$.

This variant of graph partitioning is NP-hard by a reduction from 3SAT (Hyafil and Rivest 1973) and is even hard to approximate within a constant factor (Andreev and Räcke 2006). Therefore, no efficient algorithm is currently known for solving the problem exactly, and perhaps no efficient algorithm exists. Therefore, heuristic methods are used in practice to attempt to minimize the objective function while maintaining the balancing constraint. Many methods exist for this purpose and are very successful in practice.

Although the vast majority of techniques for graph partitioning are for graphs that fit into memory on a single machine, these techniques are sufficient for many big data problems. For example, to process large datasets in parallel, a graph representing the communication required for these computations, or for the dependencies between computations, can be formed. While the data may not fit into memory, the graph may. For graphs that fit into memory on a single machine, the fastest techniques in practice include variants of the KL/FM local search algorithm (Fiduccia and Mattheyses 1982), which efficiently swap nodes between partitions and keep the assignment with lowest cut size. For balancing quality and speed, the *multilevel* approach (Hendrickson and Leland 1995; Sanders and Schulz 2011) contracts subgraphs of the graph in a coarsening phase, solves the smaller instance with a high-quality (but slow) partitioner, and uncontracts while applying local search. For high-quality partitions, methods using max-flow/min-cut are used (Sanders and Schulz 2011, 2013). Other techniques include evolutionary methods (Kim et al. 2011), diffusion (Meyerhenke and Sauerwald 2012), and spectral methods (Alpert et al. 1999). For an overview of in-memory techniques, see the survey by Buluç et al. (2016).

Though these techniques have an important place in graph partitioning, they are sparingly used to process massive datasets – those that do not fit into the working memory of a single machine.

Techniques for Massive Graphs

Recent research focuses on solving much larger instances, either by using external memory (i.e., using the disk) or in a distributed setting. These methods use variants of the *label propagation* algorithm initially proposed by Raghavan et al. (2007) for graph clustering. Label propagation is used as a parallel local search algorithm (Ugander and Backstrom 2013; Aydin et al. 2016; Slota et al. 2017), and in some approaches it is additionally used to find clusterings that are contracted in a multilevel scheme (Meyerhenke et al. 2017; Akhremtsev et al. 2015). Using these

techniques complex networks with up to a trillion edges can be partitioned.

Objective Functions

Cut Size. As discussed above, the most common objective function is to minimize the total cut size $\sum_{i < j} \omega(E_{ij})$. A related objective is to minimize the *maximum cut size* $\max_{i < j} \omega(E_{ij})$. Minimizing the cut size is useful in reducing the amount of computation required by combinatorial optimization problems, where the strategy is to subdivide the graph into blocks that are solved independently (possibly in parallel) and combine these solutions optimally (using branch and bound along the boundary vertices) or approximately by stitching together solutions across the cut edges (Lamm et al. 2017).

Communication Volume. In parallel computing, the *communication volume* is often more representative than the cut (Hendrickson and Kolda 2000). Let V_ℓ be the block containing vertex v . Then we let $D(v)$ denote the number of blocks in which vertex v has a neighbor, excluding the block containing v . That is, $D(v) = |\{j : \exists u \in V_j \neq V_\ell \text{ s.t. } \{u, v\} \in E\}|$. For a block V_i , the *total communication volume* is $\text{comm}(V_i) := \sum_{v \in V_i} c(v)D(v)$. Similar to cut size, one can minimize the *maximum communication volume* $\max_{1 \leq i \leq k} \text{comm}(V_i)$ or the *total communication volume* $\sum_{1 \leq i \leq k} \text{comm}(V_i)$. Suppose that c maps a vertex to its data size, and whole blocks of vertices are stored on a single machine, then the total communication volume represents the total amount of bandwidth needed to communicate data between adjacent blocks simultaneously, and the maximum communication volume would represent the maximum bandwidth required for communicating from a single block. Choosing the total or maximum variant of the objective function here is typically dictated by the topology of the interconnection network (Hendrickson and Kolda 2000).

Other Objectives are less common; these include the maximum degree in the *quotient graph* (the graph formed by blocks and their connections), expansion and conductance (Lang and Rao 2004), and optimizing the “shape” of partitions (Meyerhenke and Sauerwald 2012).

Edge-Based Partitioning

Edge-based partitioning is a relatively new formulation that partitions the *edges* of the input graph into k blocks of roughly the same number of edges. The objective now is to minimize the number of partitions in which each vertex is a member (Gonzalez et al. 2012). This formulation was introduced for PowerGraph (Gonzalez et al. 2012), an extension of the think-like-a-vertex (TLAV) model of computation (McCune et al. 2015) that focuses on dividing *edges* among machines, which requires vertex-centric computations to be duplicated across machines to avoid communication overhead introduced by high-degree nodes in power-law graphs. In this model, a balanced edge-based partition ensures that little communication occurs between machines containing the same vertex.

More formally, the objective is to divide the edges into k blocks $E_1, \dots, E_k \subseteq E$ such that for all $i \neq j$, $E_i \cap E_j = \emptyset$ and $\cup_{1 \leq i \leq k} E_i = E$. For each vertex v_i , denote by $A_i = \{j : \exists u \in V \text{ s.t. } \{u, v_i\} \in E_j\}$ the set of *replicas* of v_i . Then the goal is to minimize the average number of replicas $\frac{1}{|V|} \sum_{v_i \in V} |A_i|$ while satisfying the balance constraint $\max_{1 \leq i \leq k} |E_i| < (1 + \epsilon) \frac{|E|}{k}$ for some $\epsilon \geq 0$. This models the total communication between machines performing computations for that vertex.

Like other graph partitioning formulations, edge-based partitioning is NP-hard (Bourse et al. 2014), and constant factor approximations cannot be computed in polynomial time unless $P = NP$. However, Bourse et al. (2014) gave a polynomial time $O(\sqrt{\log k \log n})$ -approximation algorithm.

If the graph fits into memory, edge-based partitioning can be solved by forming a hypergraph with a node for each edge in E , and a hyperedge for each node, consisting of the edges to which it is incident. Thus, general hypergraph partitioners can be applied to this problem; however, they are more powerful than necessary, as this conversion has mostly small hyperedges.

This problem can also be solved with vertex-based graph partitioning, by creating a new graph

G' with the *split-and-connect transformation* (SPAC) of Li et al. (2017), which splits vertices and links them together, assigning weights to prevent certain edges from being cut. Then a vertex-based partition of G' gives an edge-based partition of the input graph. Combining SPAC with the kMetis partitioner gives partitions of quality comparable to hypergraph partitioning, but up to hundreds of times faster. It also gives much higher-quality partitions than streaming partitioners (discussed next); however, streaming algorithms are designed to handle massive graphs in a distributed setting.

G

Techniques for Massive Graphs

Gonzalez et al. (2012) investigate the streaming case, where edges are assigned to partitions in a single pass over the graph. They investigated randomly assigning edges to partitions, as well as a greedy strategy that prioritizes partitions that already cover a vertex incident to the edge (to avoid replicating the vertex). Their greedy method gives an average of *five* replicas per vertex for power-law graphs, and it is possible to maintain the partition information in a distributed table, giving a significant speed up over TLAV implementations. Bourse et al. (2014) later improved the replication factor by weighting each vertex by its degree. Rahimian et al. (2014) present a parallel and distributed algorithm for large graphs which is essentially a local search algorithm that iteratively improves an initial random assignment of edges to partitions.

Examples of Application

We now discuss applications of graph partitioning in the big data context.

High-Performance Computing. Scientific computing is one of the main applications and motivators for graph partitioning techniques. Typical applications in this area include lattice Boltzmann methods (Langguth et al. 2015) and simulations using the finite element method (Zhou et al. 2010) or finite volume method (Fietz et al. 2012), all of which need to partition a model of computation

and communication in order to run on a parallel computer. In this model vertices denote computation and edges represent communication. In large-scale simulations, models can contain billions of elements and simulations involve more than 100 k processors (Zhou et al. 2010). Graph partitioning is used to partition the model and then distribute the computation over a given number of processing elements while minimizing the communication overhead. For large-scale simulations, it can be helpful to choose groups of mesh nodes (Fietz et al. 2012). We very briefly outline the finite element method (leaving out technical details) and then relate it to graph partitioning.

In scientific simulations one often wants to compute an approximation to some partial differential equation that models a physical process. We use the heat equation to illustrate that. Let Ω be a region. The system that one wants to solve here is $-\Delta u = f$ in Ω with $u = 0$ on $\partial\Omega$ where $\Delta = \sum \partial_{ii}$ – the sum of the second-order derivatives. The problem asks for a function u in some function space \mathcal{V} that solves the system. However, the solution space does usually not have finite dimension. Hence, one seeks an approximation to the system. One possibility to do so is to use weak formulations of the system: Find $u \in \mathcal{V}$ such that $a(u, v) = (f, v) := \int_{\Omega} fv dx \forall v \in \mathcal{V}$ where $a(u, v) := \int_{\Omega} \nabla u \cdot \nabla v dx$ is a continuous bilinear map. In a second step, the solution space is approximated by a vector space of finite dimension, e.g., choosing $\mathcal{V}_h := \text{span}\{\phi_1, \dots, \phi_n\} \subset \mathcal{V}$ for some ϕ_i . The weak formulation now becomes: Find $u_h \in \mathcal{V}_h$ such that $a(u_h, \phi) = (f, \phi) \forall \phi \in \mathcal{V}_h$. First, since \mathcal{V}_h has finite dimension, there are $\alpha_i \in \mathbb{R}$ such that $u_h = \sum_{i=1}^n \alpha_i \phi_i$. Hence, one wants to find α_i such that $a(\sum_{i=1}^n \alpha_i \phi_i, \phi) = (f, \phi) \forall \phi \in \mathcal{V}_h$ which is easily rewritten to $\sum_{i=1}^n \alpha_i a(\phi_i, \phi_j) = (f, \phi_j) \forall j \in \{1, \dots, n\}$. This however means that one can solve a linear equation system $Ax = b$ with $a_{ij} = a(\phi_i, \phi_j)$ and $b_i = (f, \phi_i)$. Also note that due to the approximation that is done here, the output solution may differ from the real solution, e.g., there are errors involved. Roughly speaking, in practice one increases n to reduce

the error which is why the linear equation systems become very large. However, typically the arising equation systems are sparse and solved using an iterative method such as the Jacobi method which requires sparse matrix-vector multiplication as a core component. In essence, at each time step, one has an approximate solution x_k to the linear equation system and computes a new approximation x^{k+1} using the equation $x^{k+1} = D^{-1}(b - Rx^k)$ where $D = \text{diag}(A)$, $R = A - D$. The process is repeated until the residual error is small. One approach to parallelize the computation is to distribute columns of A and x over equally processors such that communication is minimized. To build our model of computation and communication, one rewrites the iterative formula as $x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^k)$.

It is now easy to define a graph/model $G = (\{1, \dots, n\}, E)$ which has one node per column of the matrix and edges such that $e = \{i, j\} \in E \Leftrightarrow a_{i,j} \neq 0$. Since the amount of computation for a node is proportional to its degree, one uses vertex degrees as vertex weight. Additionally, one associates $x_i^k, x_i^{k+1}, A_{i,*}$ with node $i \in V$. After partitioning the model, PE l obtains columns associated with block V_l , stores $x_i^k, x_i^{k+1}, A_{i,*}$, and computes x_i^{k+1} for $i \in V_l$.

Decades ago, minimizing edge cut was the way to go since in meshes it is highly correlated with communication volume. For networks with complex structure, it is more important to optimize the total/maximum communication volume (Buluç and Madduri 2012).

Scalable Distributed Systems. Large companies that offer web services to users like Amazon, Facebook, Google, or Twitter face the problem of distributed system design. Assigning components in a distributed system has a significant impact on performance, as well as resource usage. A problem that often arises is to assign users to servers or data records to storage units (Tran et al. 2012). This is a prototypical big data application as the underlying networks that need partitioning often contain billions of vertices and trillions of edges (Shalita et al. 2016). Shalita et al. (2016) disclose two concrete applications used at Facebook: HTTP request

routing optimization and storage sharding optimization.

For example, in HTTP request routing, HTTP requests have to be assigned to compute clusters. If a HTTP request arrives, any required data to serve the request has to be fetched from external storage servers and will be cached in a local main memory-based cache. Intuitively, it is clear that HTTP requests that request similar data should be assigned to the same compute cluster to use the main memory-based caches efficiently. As friends and socially similar users tend to consume the same data, the static assignment is done by partitioning a graph where vertices represent users and edges represent friendships between them. Here, the edge cut is used as the optimization objective.

Large-Scale Biology Applications. There are numerous big data applications in biology that use graph partitioning techniques, for example, 3D cell nuclei segmentation and sequencing datasets for parallel genomics analyses.

Recent advances enabled researches to do in vivo investigation of biological structures. The instances can contain images with total size of about 10 TB per embryo (Tomer et al. 2012). This yields demand for automated methods to process these images. Arz et al. (2017) use graph partitioning to split the connected components of a binarized image recursively. The graph model of a connected component is built by setting foreground voxels to be vertices and inserting edges between neighboring cells. Edge weights depend on the likelihood that vertices belong together.

Nowadays, high-throughput sequencing machines allow researchers to generate massive numbers of short genomic fragments (reads) per run. Sequencing has become an important technology for a wide range of applications. In many of these applications, mapping sequenced reads to their potential genomic origin is the first fundamental step for subsequent analyses. The large amounts of data generated from such machines present significant computational challenges for analysis. Jammula et al. (2017) present a parallel algorithm for partitioning

large-scale read datasets in order to facilitate distributed-memory parallel analyses. Roughly speaking, a de Bruijn graph represents overlaps between the reads that have been computed by the sequencing machine. The algorithm works as follows: build a de Bruijn graph from the given set of input reads, compact it to reduce its size, partition the resulting graph using a standard parallel graph partitioning algorithm, and then generate a partition of the read dataset from the partition of the de Bruijn graph. This enables further analyses that run on the distributed memory machine to run efficiently.

G

Cross-References

- ▶ [Graph Processing Frameworks](#)
- ▶ [Large-Scale Graph Processing System](#)
- ▶ [Parallel Graph Processing](#)

References

- Akhremtsev Y, Sanders P, Schulz C (2015) (Semi-)external algorithms for graph partitioning and clustering. In: Proceeding of 17th workshop on algorithm engineering and experiments (ALENEX 2015). SIAM, pp 33–43
- Alpert CJ, Kahng AB, Yao SZ (1999) Spectral partitioning with multiple eigenvectors. *Discret Appl Math* 90(1):3–26
- Andreev K, Räcke H (2006) Balanced graph partitioning. *Theory Comput Syst* 39(6):929–939
- Arz J, Sanders P, Stegmaier J, Mikut R (2017) 3D cell nuclei segmentation with balanced graph partitioning. *CoRR* abs/1702.05413
- Aydin K, Bateni M, Mirrokni V (2016) Distributed balanced partitioning via linear embedding. In: Proceeding of the ninth ACM international conference on web search and data mining. ACM, pp 387–396
- Bichot C, Siarry P (eds) (2011) Graph partitioning. Wiley, London
- Bourse F, Lelarge M, Vojnovic M (2014) Balanced graph edge partition. In: Proceeding 20th ACM SIGKDD international conference on knowledge discovery and data mining, KDD'14. ACM, pp 1456–1465
- Buluç A, Madduri K (2012) Graph partitioning for scalable distributed graph computations. In: Proceeding of 10th DIMACS implementation challenge, contemporary mathematics. AMS, pp 83–102
- Buluç A, Meyerhenke H, Safro I, Sanders P, Schulz C (2016) Recent advances in graph partitioning. In:

- Kliemann L, Sanders P (eds) Algorithm engineering. Springer, Cham, pp 117–158
- Fiduccia CM, Mattheyses RM (1982) A linear-time heuristic for improving network partitions. In: Proceedings of the 19th conference on design automation, pp 175–181
- Fietz J, Krause M, Schulz C, Sanders P, Heuveline V (2012) Optimized hybrid parallel lattice Boltzmann fluid flow simulations on complex geometries. In: Proceeding of Euro-Par 2012 parallel processing. LNCS, vol 7484. Springer, pp 818–829
- Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) PowerGraph: distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th USENIX symposium on operating systems design and implementation (OSDI 12), USENIX, pp 17–30
- Hendrickson B, Kolda TG (2000) Graph partitioning models for parallel computing. *Parallel Comput* 26(12):1519–1534
- Hendrickson B, Leland R (1995) A multilevel algorithm for partitioning graphs. In: Proceeding of the ACM/IEEE conference on supercomputing'95. ACM
- Hyafil L, Rivest R (1973) Graph partitioning and constructing optimal decision trees are polynomial complete problems. Technical report 33, IRIA – Laboratoire de Recherche en Informatique et Automatique
- Jammula N, Chockalingam SP, Aluru S (2017) Distributed memory partitioning of high-throughput sequencing datasets for enabling parallel genomics analyses. In: Proceeding of 8th ACM international conference on bioinformatics, computational biology, and health informatics, ACM-BCB'17. ACM, pp 417–424
- Karypis G, Aggarwal R, Kumar V, Shekhar S (1999) Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Trans Very Large Scale Integr (VLSI) Syst* 7(1):69–79
- Kim J, Hwang I, Kim YH, Moon BR (2011) Genetic approaches for graph partitioning: a survey. In: 13th genetic and evolutionary computation (GECCO). ACM, pp 473–480
- Lamm S, Sanders P, Schulz C, Strash D, Werneck RF (2017) Finding near-optimal independent sets at scale. *J Heuristics* 23(4):207–229
- Lang K, Rao S (2004) A flow-based method for improving the expansion or conductance of graph cuts. In: Proceedings of the 10th international integer programming and combinatorial optimization conference. LNCS, vol 3064. Springer, pp 383–400
- Langguth J, Sourouri M, Lines GT, Baden SB, Cai X (2015) Scalable heterogeneous CPU-GPU computations for unstructured tetrahedral meshes. *IEEE Micro* 35(4):6–15
- Li L, Geda R, Hayes AB, Chen Y, Chaudhari P, Zhang EZ, Szegedy M (2017) A simple yet effective balanced edge partition model for parallel computing. *Proc ACM Meas Anal Comput Syst* 1(1):14:1–14:21
- McCune RR, Weninger T, Madey G (2015) Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Comput Surv* 48(2):25:1–25:39
- Meyerhenke H, Sauerwald T (2012) Beyond good partition shapes: an analysis of diffusive graph partitioning. *Algorithmica* 64(3):329–361
- Meyerhenke H, Sanders P, Schulz C (2017) Parallel graph partitioning for complex networks. *IEEE Trans Parallel Distrib Syst* 28:2625–2638
- Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. *Phys Rev E* 76(3):036106
- Rahimian F, Payberah AH, Girdzijauskas S, Haridi S (2014) Distributed vertex-cut partitioning. In: IFIP international conference on distributed applications and interoperable systems. Springer, pp 186–200
- Sanders P, Schulz C (2011) Engineering multilevel graph partitioning algorithms. In: Proceedings of the 19th European symposium on algorithms. LNCS, vol 6942. Springer, pp 469–480
- Sanders P, Schulz C (2013) High quality graph partitioning. In: Proceedings of the 10th DIMACS implementation challenge – graph clustering and graph partitioning. AMS, pp 1–17
- Schloegl K, Karypis G, Kumar V (2003) Graph partitioning for high-performance scientific simulations. In: Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L, White A (eds) Sourcebook of parallel computing. Morgan Kaufmann Publishers, San Francisco, pp 491–541
- Shalita A, Karrer B, Kabiljo I, Sharma A, Presta A, Adcock A, Kllapi H, Stumm M (2016) Social hash: an assignment framework for optimizing distributed systems operations on social networks. In: Argyraki KJ, Isaacs R (eds) 13th USENIX symposium on networked systems design and implementation, NSDI. USENIX Association, pp 455–468
- Slota GM, Rajamanickam S, Devine K, Madduri K (2017) Partitioning trillion-edge graphs in minutes. In: Proceedings of the 31st IEEE international parallel and distributed processing symposium (IPDPS 2017), pp 646–655
- Tomer R, Khairy K, Amat F, Keller PJ (2012) Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy. *Nat Methods* 9(7):755–763
- Tran DA, Nguyen K, Pham C (2012) S-clone: socially-aware data replication for social networks. *Comput Netw* 56(7):2001–2013
- Ugander J, Backstrom L (2013) Balanced label propagation for partitioning massive graphs. In: Proceedings of the sixth ACM international conference on web search and data mining, WSDM'13. ACM, pp 507–516
- Zhou M, Sahni O, Devine KD, Shephard MS, Jansen KE (2010) Controlling unstructured mesh partitions for massively parallel simulations. *SIAM J Sci Comput* 32(6):3201–3227

Graph Path Navigation

Marcelo Arenas¹, Pablo Barceló², and Leonid Libkin³

¹Pontificia Universidad Católica de Chile, Santiago, Chile

²Universidad de Chile, Santiago, Chile

³School of Informatics, University of Edinburgh, Edinburgh, UK

Synonyms

Navigational queries; Path queries; Regular path queries

Definitions

Navigational query languages for graph databases allow to recursively traverse the edges of a graph while checking for the existence of a path that satisfies certain regular conditions. The basic building block of such languages is the class of *regular path queries* (RPQs), which are expressions that compute the pairs of nodes that are linked by a path whose label satisfies a regular expression. RPQs are often extended with features that turn them more flexible for practical applications, e.g., with the ability to traverse edges in the backward direction (RPQs with *inverses*) or to express arbitrary patterns over the data (*conjunctive* RPQs).

Overview

Graph Databases

Graph databases provide a natural encoding of many types of data where one needs to deal with objects and relationships between them. An object is represented as a node, and a relationship between two objects is represented as an edge, where labels are assigned to these edges to indicate what types of relationships they represent. This interpretation of data often arises in situations where one needs to *navigate* in the graph

and reason about the *paths* in it. To talk about such navigation, we can abstract graph databases as edge-labeled graphs. Formally, assuming that \mathbf{L} is a fixed infinite set of edge labels, we have the following formal definition of a graph database.

Definition 1 A graph database G is a pair (N, E) , where:

1. N is a finite set of *nodes*; and
2. E is a finite set of *labeled edges*, that is, a finite subset of $V \times \mathbf{L} \times V$. \square

Example 1 Figure 1 shows a simple graph database $G = (N, E)$ storing information about people and their relationships. Here N has four nodes:

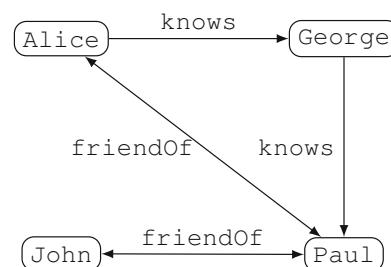
{Alice, George, John, Paul}

and E contains six labeled edges:

$\{(Alice, \text{knows}, George),$
 $(George, \text{knows}, Paul),$
 $(Alice, \text{friendOf}, Paul),$
 $(Paul, \text{friendOf}, Alice),$
 $(John, \text{friendOf}, Paul),$
 $(Paul, \text{friendOf}, John)\}$. \square

Path Queries

The simplest type of query on a graph database extracts nodes according to the relationship between them. We assume that \mathbf{V} is an infinite set



Graph Path Navigation, Fig. 1 A graph database

of variables that is disjoint with \mathbf{L} . Then given $x, y \in \mathbf{V}$ and $\ell \in \mathbf{L}$, the following is a basic query

$$x \xrightarrow{\ell} y$$

that asks for the pairs of nodes in a graph database connected by an edge-labeled ℓ . The semantics of a such a query Q on a graph database $G = (N, E)$, denoted by $\llbracket Q \rrbracket_G$, is the set of *matchings* $h : \{x, y\} \rightarrow N$ such that $(h(x), \ell, h(y)) \in E$.

Example 2 Evaluating the basic query $Q_1 = x \xrightarrow{\text{knows}} y$ over the graph database G in Fig. 1 produces as a result:

	x	y
h_1	Alice	George
h_2	George	Paul

As depicted in the table, we have that $\llbracket Q_1 \rrbracket_G = \{h_1, h_2\}$, where $h_1(x) = \text{Alice}$, $h_1(y) = \text{George}$, $h_2(x) = \text{George}$, and $h_2(y) = \text{Paul}$.

□

While basic queries like the one above allow for extracting valuable information from a graph database, it is often useful to provide more flexible querying mechanisms that allow to *navigate* the topology of the data. One example of such a query is to find all friends-of-a-friend of some person in a social network. We are not only interested in immediate acquaintances of a person but also in people he/she might know through other people, namely, his/her friends-of-a-friend, their friends, and so on. For instance, in the graph in Fig. 1, we may want to discover that Alice is connected with John through a common friend.

Queries such as the one above are called *path queries*, since they require navigation in a graph using paths of arbitrary lengths. Path queries have found applications in areas such as the Semantic Web (Alkhateeb et al. 2009; Pérez et al. 2010; Paths 2009), provenance (Holland et al. 2008), and route-finding systems (Barrett et al. 2000), among others. In this entry, we provide a brief

overview of path queries and some of their extensions; for detailed surveys, the reader is referred to Barceló (2013) and Angles et al. (2017).

Regular Path Queries

A *path* p in a graph database G is a sequence e_1, e_2, \dots, e_n of edges of the form $e_i = (a_i, \ell_i, b_i)$ such that $n \geq 0$ and $b_j = a_{j+1}$ for every $j \in \{1, \dots, n-1\}$. That is, each edge in a path starts in the node where the previous edge ends. If $n = 0$ we refer to p as the empty path. We write *labels*(p) for the string of labels formed by the sequence of edges in p , that is, *labels*(p) = ε if $n = 0$ and *labels*(p) = $\ell_1 \ell_2 \dots \ell_n$ otherwise. Finally, we say that a_1 is the starting node of p and b_n is the ending node of p .

Example 3 $p = (\text{Alice}, \text{knows}, \text{George}), (\text{George}, \text{knows}, \text{Paul})$ is a path in the graph database in Fig. 1. This path p is a path of length 2, its starting node is Alice, its ending node is Paul, and *labels*(p) = knows knows. □

Definition 2 A *regular path query* (RPQ) Q is an expression of the form

$$x \xrightarrow{r} y,$$

where $x, y \in \mathbf{V}$ and r is a regular expression over \mathbf{L} . The evaluation of Q over a graph database $G = (N, E)$, denoted by $\llbracket Q \rrbracket_G$, is the set of matchings $h : \{x, y\} \rightarrow N$ for which there exists a path p in G whose starting node is $h(x)$, ending node is $h(y)$, and *labels*(p) is a string in the regular language defined by r . □

Thus, an RPQ $x \xrightarrow{r} y$ asks for the pair of nodes in a graph database that are the endpoints of a path whose labels conform to the regular expression r .

Example 4 Assume that Q_2 is the RPQ $x \xrightarrow{\text{knows}^+} y$, where knows^+ is a regular expression that defines the language of nonempty strings of the form $\text{knows } \text{knows } \dots \text{knows}$. The following is the result of evaluating Q_2 over the graph database G shown in Fig. 1:

	x	y
h_1	Alice	George
h_2	George	Paul
h_3	Alice	Paul

In this case, $h_3 \in \llbracket Q_2 \rrbracket_G$ since for the path $p = (\text{Alice}, \text{knows}, \text{George}), (\text{George}, \text{knows}, \text{Paul})$ in G , we have that the starting node of p is Alice, the ending node of p is Paul, $\text{labels}(p) = \text{knows} \text{ knows}$, and $\text{knows} \text{ knows}$ is a string in the regular language defined by knows^+ . \square

Notice that Definition 2 does not impose the restriction that x and y be distinct variables, so an RPQ of the form $x \xrightarrow{r} x$ is valid, and its semantics is well defined. Such an RPQ asks for cycles whose labels conform to the regular expression r .

Constant values are usually allowed in RPQs. For the sake of readability, we do not consider them separately, as the syntax and semantics of an RPQ with constants are defined exactly in the same way as in Definition 2. We only give an example of such a query to illustrate how constants are used in RPQs.

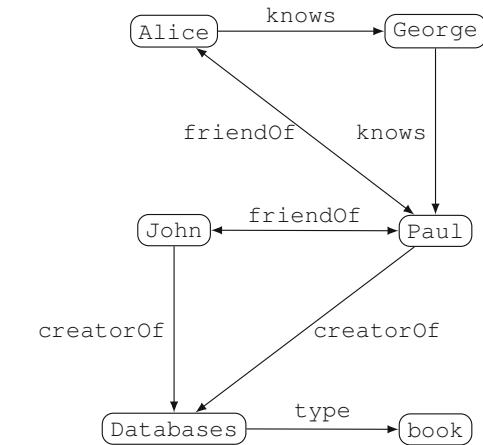
Example 5 Assume that Q_3 is the RPQ $\text{Alice} \xrightarrow{\text{knows}^+} z$, where z is a variable and Alice is a constant representing a node in a graph. The following is the result of evaluating Q_3 over the graph database shown in Fig. 1:

	z
h_4	George
h_5	Paul

\square

Regular Path Queries with Inverse

Let G be the graph database shown in Fig. 2, which is an extension of our running example with information about creations. Assume that we want to retrieve pairs of people that are co-creators of the same artifact. We can think of this query as a path in G : starting from a person, we traverse an edge with label `creatorOf` to reach an artifact, and then from there we traverse again an edge with label `creatorOf` to reach a person,



Graph Path Navigation, Fig. 2 A graph database including information about creations

who is then another creator of the artifact. But there is an issue with such a path as the second edge with label `creatorOf` has to be traversed in the opposite direction.

To overcome this problem, RPQs are often extended with the ability to traverse edges in both directions, which gives rise to the notion of RPQ with *inverses* (Calvanese et al. 2000). To define this, let \mathbf{L}^\pm be the extension of \mathbf{L} with the symbol ℓ^- , for each $\ell \in \mathbf{L}$. Moreover, for a graph database $G = (N, E)$, let G^\pm be the extension of G by adding the edges (b, ℓ^-, a) , for each $(a, \ell, b) \in E$.

Definition 3 A *regular path query with inverses* (2RPQ) Q is an expression of the form

$$x \xrightarrow{r} y,$$

where $x, y \in \mathbf{V}$ and r is a regular expression over \mathbf{L}^\pm . The evaluation of Q over a graph database G , denoted by $\llbracket Q \rrbracket_G$, is defined as $\llbracket Q \rrbracket_{G^\pm}$. \square

In this definition, $\llbracket Q \rrbracket_{G^\pm}$ is well defined as Q is an RPQ over G^\pm .

Example 6 The following 2RPQ Q_4 can be used to retrieve pairs of people that are co-creators of the same artifact in the graph database shown in Fig. 2:

$$x \xrightarrow{\text{creatorOf creatorOf}^-} y$$

In particular, if h_6 is a matching such that $h_6(x) = \text{John}$ and $h_6(y) = \text{Paul}$, then we have that $h_6 \in \llbracket Q \rrbracket_G$ as $p = (\text{John}, \text{creatorOf}, \text{Databases})$, $(\text{Databases}, \text{creatorOf}^-, \text{Paul})$ is a path in G^\pm such that $\text{labels}(p)$ is a string that conforms to the regular expression `creatorOf creatorOf^-`.

Notice that we can retrieve pairs of people who are connected by a co-creation path of arbitrary length by using the following 2RPQ:

$$x \xrightarrow{(\text{creatorOf creatorOf}^-)^+} y$$

□

Conjunctive Regular Path Queries

A more expressive form of graph database queries is obtained by viewing RPQs as basic building blocks and then defining conjunctive queries over them. Recall that conjunctive queries, over relational databases, are obtained by closing relational atoms under conjunction and existential quantification (or, equivalently, under selection, projection, and join operations of relational algebra). When the same operations are applied to RPQs, they give rise to the notion of conjunctive RPQs.

Definition 4 A *conjunctive regular path query* (CRPQ) is an expression of the form

$$\text{ans}(u_1, \dots, u_k) :- \bigwedge_{i=1}^n v_i \xrightarrow{r_i} w_i \quad (1)$$

where each $v_i \xrightarrow{r_i} w_i$, for $i \in \{1, \dots, n\}$, is an RPQ that may include constants and u_1, \dots, u_k is a sequence of pairwise distinct variables among those that occur as v_i s and w_i s. □

The left-hand side of (1) is called the head of the query, and its right-hand side is called the body. Variables in the body of (1) need not be pairwise distinct; in fact, sharing variables allows joining results of the RPQs in the body

of the query, which is a fundamental feature of CRPQs. As in the case of relational conjunctive queries, variables mentioned in the body of (1) but not in the head (i.e., those that are not in the answer) are assumed to be existentially quantified (in other words, projected away). Finally, if $k = 0$, then (1) is called a Boolean CRPQ, as its evaluation results in either a set containing a matching with empty domain or an empty set of matchings, which represent the values true and false, respectively.

The semantics of (1) is defined as follows. Assume that $G = (N, E)$ is a graph database, and let x_1, \dots, x_m be the variables occurring in the body of (1). Then a matching $h : \{x_1, \dots, x_m\} \rightarrow N$ is said to satisfy the body of (1) if for every $i \in \{1, \dots, n\}$:

$$h|_{\{v_i, w_i\} \cap V} \in \llbracket v_i \xrightarrow{r_i} w_i \rrbracket_G,$$

where $h|_{\{v_i, w_i\} \cap V}$ is the restriction of matching h to the domain $\{v_i, w_i\} \cap V$. A matching $g : \{u_1, \dots, u_k\} \rightarrow N$ is an *answer* to (1) over G if there is an extension $h : \{x_1, \dots, x_m\} \rightarrow N$ of g that satisfies the body of (1). The evaluation of a CRPQ Q over G , written $\llbracket Q \rrbracket_G$, is the set of answers to Q over G .

Example 7 Let $G = (N, E)$ be the graph database in Fig. 2. Then the following CRPQ Q_5 can be used to retrieve the pairs of people that are co-authors of the same book:

$$\begin{aligned} \text{ans}(x, y) :- & x \xrightarrow{\text{creatorOf}} z \wedge \\ & y \xrightarrow{\text{creatorOf}} z \wedge \\ & z \xrightarrow{\text{type}} \text{book} \end{aligned}$$

Let $g : \{x, y\} \rightarrow N$ and $h : \{x, y, z\} \rightarrow N$ be matchings such that $g(x) = \text{John}$, $g(y) = \text{Paul}$, $h(x) = \text{John}$, $h(y) = \text{Paul}$, and $h(z) = \text{Databases}$. We have that $g \in \llbracket Q_5 \rrbracket_G$ since $g = h|_{\{x, y\}}$, $h|_{\{x, z\}} \in \llbracket x \xrightarrow{\text{creatorOf}} z \rrbracket_G$, $h|_{\{y, z\}} \in \llbracket y \xrightarrow{\text{creatorOf}} z \rrbracket_G$, and $h|_{\{z\}} \in \llbracket z \xrightarrow{\text{type}} \text{book} \rrbracket_G$. □

Key Research Findings

The Complexity of Evaluating Path Queries

We now consider the cost of evaluating a query in a graph query language \mathcal{L} . More precisely, the complexity of the problem \mathcal{L} -EVAL is defined as follows: Given a graph database G , a query $Q \in \mathcal{L}$, and a matching h , does h belong to the evaluation of Q over G ? (in symbols, is $h \in \llbracket Q \rrbracket_G$?). Notice that the input to \mathcal{L} -EVAL consists of a graph database G , a query Q and a matching h , and, thus, it measures the *combined complexity* of the evaluation problem. Since the size of h is bounded by the size of Q and the size of G , it suffices to measure the combined complexity in terms of $|Q|$ and $|G|$, the sizes of Q and G .

In practice, queries are usually much smaller than graph databases, so one is also interested in the *data complexity* of the evaluation problem, which is measured only in terms of the size of the graph database (i.e., the query is assumed to be fixed).

The combined and data complexities of RPQ-EVAL, 2RPQ-EVAL, and CRPQ-EVAL are shown in Fig. 3.

When we state that data complexity is NLOGSPACE-complete, we mean that for each fixed query Q in those language, the evaluation problem can be solved in NLOGSPACE, and there are some queries – in fact, RPQs – for which it is NLOGSPACE-hard.

We now outline the key ideas behind the $O(|G| \cdot |Q|)$ algorithm as well as NLOGSPACE-completeness and NP-completeness.

Given a database graph $G = (N, E)$, a 2RPQ $Q = x \xrightarrow{r} y$, and a matching $h : \{x, y\} \rightarrow N$, we show that 2RPQ-EVAL can be solved in time $O(|G| \cdot |Q|)$. The idea is from Mendelzon and Wood (1995). Assume that $h(x) = a$ and $h(y) = b$. First, we compute G^\pm from G in time $O(|G|)$, and then we compute in time $O(|Q|)$, a nonde-

terministic finite automaton with ε -transitions (ε -NFA) A_r , that defines the same regular language as r . Let $G^\pm(a, b)$ be the NFA obtained from G^\pm by setting its initial and final states to be a and b , respectively. We know that $h \in \llbracket Q \rrbracket_G$ if and only if $h \in \llbracket Q \rrbracket_{G^\pm}$, and the latter is equivalent to checking whether there exists a word accepted by both $G^\pm(a, b)$ and A_r . In turn, this is equivalent to checking the product of $G^\pm(a, b)$ and A_r for nonemptiness, which can be done in linear time in the size of the product automaton, i.e., in time $O(|G^\pm| \cdot |A_r|)$. The whole process takes time $O(|G| + |Q| + |G^\pm| \cdot |A_r|)$, that is, $O(|G| \cdot |Q|)$.

If the query Q is fixed, the same algorithm can be performed in NLOGSPACE. Indeed, we just need to compute reachability in the product automaton, and reachability in graphs is computable in NLOGSPACE. Of course the whole product requires more than logarithmic space, but it need not be built, as reachability can be checked by using standard on the fly techniques. Moreover, reachability in graphs is known to be NLOGSPACE-complete, so even RPQ query evaluation can be NLOGSPACE-complete in data complexity.

Finally, to see that CRPQs can be evaluated in NP in combined complexity, it just suffices to guess the values of all the v_i s and w_j s and then check in polynomial time that all the RPQs from (1) hold. And since the evaluation problem for relational conjunctive queries over graphs is NP-hard in data complexity, so is the evaluation problem for the more expressive CRPQs.

Simple Path Semantics

The evaluation of RPQs as defined here is based on a semantics of arbitrary paths. It has been argued, on the other hand, that for some applications it is more reasonable to apply a semantics based on *simple paths* only, i.e., those without repeating nodes. Under such a semantics, an RPQ $x \xrightarrow{r} y$ computes the pairs of nodes that are

Graph Path Navigation,

Fig. 3 The complexity of the evaluation problem for path queries

	Combined complexity	Data complexity
RPQ-EVAL	$O(G \cdot Q)$	NLOGSPACE-complete
2RPQ-EVAL	$O(G \cdot Q)$	NLOGSPACE-complete
CRPQ-EVAL	NP-complete	NLOGSPACE-complete

linked by a *simple path* whose label satisfies r . However, evaluation of RPQs under the simple path semantics becomes NP-complete even in data complexity (Mendelzon and Wood 1995).

While this casts doubt on its practical applicability, a closely related semantics was implemented in a practical graph DBMS, namely, Neo4j (Robinson et al. 2013). There, paths in graph patterns can only be matched by paths that do not have repeated *edges*. While still NP-complete in data complexity in the worst case, in practice this semantics could behave well, as worst-case scenarios do not frequently occur in real life.

Extensions

(C)RPQs have been extended with several features. One extension is a *nesting* operator that allows one to perform existential tests over the nodes of a path, similarly to what is done in the XML navigational language XPath (Pérez et al. 2010). Another extension is the ability to compare paths (Barceló et al. 2012). The idea is that paths are named in an RPQ, like $x \xrightarrow{\pi:r} y$, and then the name π can be used in the query. Without defining them formally, we give an example:

$$\text{ans}(x, y) :- x \xrightarrow{p_1:a^*} z \wedge z \xrightarrow{p_2:b^*} y \wedge \text{equal_length}(p_1, p_2)$$

says that there is a path p_1 from x to z labeled with as and a path p_2 from z to y labeled with bs and these paths have equal lengths (expressed by the predicate $\text{equal_length}(p_1, p_2)$). The equal length predicate belongs to the well-known class of regular relations, but the above query says that there is a path from x to y whose label is of the form $a^n b^n$ for some n . This of course is not a regular property, even though we only used regular languages and relations in the query. Such extended CRPQs still behave well: their data complexity remains in NLOGSPACE, and their combined complexity goes up to PSPACE, which is exactly the combined complexity of relational algebra. However such additions must be handled with care: if we add common predicates

on paths such as subsequence or subword, query evaluation becomes completely infeasible or even undecidable. We refer to Barceló et al. (2012) and Barceló (2013) for further discussions.

The last extension is constituted by adding mechanisms for checking regular properties over an extended data model in which each node carries data values from an infinite domain. This models graph databases as they occur in real life, namely, *property graphs*, where nodes and edges can carry tuples of key-value pairs. Extending RPQs and CRPQs requires choosing a proper analog of regular expressions. There are many proposals, but with few exceptions they lead to very high complexity of query evaluation. The one that maintains good complexity is given by *register automata*. Such extensions still have NLOGSPACE data complexity and PSPACE combined complexity and come with different syntactic ways of describing navigation that mixes data and topology of the graph. We refer the reader to Libkin et al. (2016) for details and the surveys by Barceló (2013) and Angles et al. (2017) for a thorough review of the literature on extensions of CRPQs.

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)
- ▶ [Graph Query Processing](#)

References

- Alkhateeb F, Baget J, Euzenat J (2009) Extending SPARQL with regular expression patterns (for querying RDF). *J Web Sem* 7(2):57–73
- Angles R, Arenas M, Barceló P, Hogan A, Reutter JL, Vrgoc D (2017) Foundations of Modern Graph Query Languages. *ACM Computing Surveys* 50(5)
- Barceló P (2013) Querying graph databases. In: Proceedings of the 32nd ACM Symposium on Principles of Database Systems, PODS 2013, pp 175–188
- Barceló P, Libkin L, Lin AW, Wood PT (2012) Expressive languages for path queries over graph-structured data. *ACM Trans Database Syst* 37(4):31
- Barrett CL, Jacob R, Marathe MV (2000) Formal-language-constrained path problems. *SIAM Journal on Computing* 30(3):809–837

- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2000) Containment of conjunctive regular path queries with inverse. In: Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning, KR 2000, pp 176–185
- Holland DA, Braun UJ, Maclean D, Muniswamy-Reddy KK, Seltzer MI (2008) Choosing a data model and query language for provenance. In: 2nd International Provenance and Annotation Workshop (IPA)
- Libkin L, Martens W, Vrgoč D (2016) Querying graphs with data. J ACM 63(2):14
- Mendelzon AO, Wood PT (1995) Finding regular simple paths in graph databases. SIAM J Comput 24(6):1235–1258
- Paths TFP (2009) Use cases in property paths task force. http://www.w3.org/2009/sparql/wiki/TaskForce:PropertyPaths#Use_Cases
- Pérez J, Arenas M, Gutierrez C (2010) nSPARQL: A navigational language for RDF. J Web Sem 8(4): 255–270
- Robinson I, Webber J, Eifrem E (2013) Graph Databases, 1st edn. O'Reilly Media

Graph Pattern Matching

Yinghui Wu¹ and Arijit Khan²

¹Washington State University, Pullman,
WA, USA

²Nanyang Technological University, Singapore,
Singapore

Definitions

The graph pattern matching problem is to find the answers $Q(G)$ of a pattern query Q in a given graph G . The answers are induced by specific query language and ranked by a quality measure. The problem can be categorized into three classes (Khan and Ranu 2017): (1) Subgraph/supergraph containment query, (2) graph similarity queries, and (3) graph pattern matching.

In the context of searching a graph database D that consists of many (small) graph transactions, the graph pattern matching finds the answers $Q(G)$ as a set of graphs from D . For subgraph (resp. supergraph containment) query, it is to find $Q(G)$ that are subgraphs (resp. supergraphs) of Q . The graph similarity queries are to find $Q(G)$

as all graph transactions that are similar to Q for a particular similarity measure.

In the context of searching a single graph G , graph pattern matching is to find all the occurrences of a query graph Q in a given data graph G , specified by a matching function. The remainder of this entry discusses various aspects of graph pattern matching in a single graph.

Overview

G

This entry discusses foundations of graph pattern matching problem: formal definitions, query languages, and complexity bounds.

Data graph. A data graph $G = (V, E, L)$ is a directed graph with a finite set of nodes V , a set of edges $E \subseteq V \times V$, and a function L that assigns node and edge content. Given a query $Q = (V_Q, E_Q, L_Q)$, a node $v \in V$ in G with content $L(v)$ (resp. edge $e \in E$ with content $L(e)$) that satisfies the constraint $L(u)$ of a node $u \in V_Q$ (resp. $L(e_u)$ of an edge $e_u \in E_Q$) is a candidate match of v (resp. e). Common data graph models include RDF and property graphs (see chapter “► Graph Data Models”).

Graph pattern query. A graph pattern query Q is a directed graph (V_Q, E_Q, L_Q) that consists of a set of nodes V_Q and edges E_Q , as defined for data graphs. For each node $v \in V_Q$ (resp. $e \in E_Q$), a function L_Q assigns a predicate $L(v)$ (resp. $L(e)$) for each v (resp. e). The predicate $L(\cdot)$ can be expressed as conjunction of atomic formulas of attributes from node and edge schema in a data graph or information retrieval metrics.

Graph pattern matching. A match $Q(G)$ of a pattern query Q in a data graph G is a substructure of G induced by a matching function $F \subseteq V_Q \times V$. The node $F(u)$ (respectively, $F(e)$) is called a match of a node $u \in V_Q$ (respectively, $e \in E_Q$). Given a query $Q = (V_Q, E_Q, L_Q)$, a data graph $G = (V, E, L)$, the graph pattern matching problem is to find all the matches of Q in G .

Subgraph pattern queries. Traditional graph pattern queries are defined by subgraph isomorphism (Ullmann 1976): (1) each node in Q has a unique label, and (2) the matching function F specifies a subgraph isomorphism, which is an injective function that enforces an edge-to-edge matching and label equality.

The subgraph isomorphism problem is NP complete. Ullmann proposed a first practical backtracking algorithm for subgraph isomorphism search in graphs (Ullmann 1976). The algorithm enumerates exact matches by incrementing partial solutions and prunes partial solutions when it determines that they cannot be completed. A number of methods such as VF2 (Cordella et al. 2004), QuickSI (Shang et al. 2008), GraphQL (He and Singh 2008), GADDI (Zhang et al. 2009), and SPATH (Zhao and Han 2010) are proposed for graph pattern queries defined by subgraph isomorphism. These algorithms follow the backtracking principle as in (Ullmann 1976) and improve the efficiency by exploiting different join orders and various pruning techniques. Lee et al. conducted an experimental study (Lee et al. 2012) to compare these methods, and experimentally verified that QuickSI designed for handling small graphs often outperforms its peers for both small and large graphs.

The major challenge for querying big graphs is twofold: (1) The query models have high complexity. For example, subgraph isomorphism is NP hard. (2) It is often hard for end users to write accurate graph pattern queries. Major research effort has been focused on two aspects: (a) developing computationally efficient query models and algorithms, and (b) user-friendly graph search paradigms.

Key Research Findings

Computationally Efficient Query Models

Graph simulation. The class of simulation-based query models includes graph simulation and bounded simulation (Fan et al. 2010), strong simulation (Ma et al. 2011), and pattern queries

with regular expressions (Fan et al. 2011). A graph simulation $R \subseteq V_Q \times V$ relaxes subgraph isomorphism to an edge-to-edge matching relation. For any node $(u, v) \in R$, u and v have the same label, and each child u' of u must have at least a match v' as a child of v , such that $(u', v') \in R$. Strong simulation (Ma et al. 2011) enforces simulation for both parents and children of a node in Q . Regular expressions are posed on edges in Q to find paths with concatenated edge labels satisfying the regular expression (Fan et al. 2011). All these query models can be evaluated with low polynomial time.

Approximate pattern matching. Approximate graph matching techniques make use of cost functions to find approximate ($\text{top-}k$) matches with fast solutions. The answer quality is usually characterized by aggregated proximity among matches determined by their neighborhood, which is more effective to cope with ambiguous queries with possible mismatches in data graphs. Traditional cost measures include graph edit distance, maximum common subgraph, number of mismatched edges, and various graph kernels. Index techniques and synopses are typically used to improve the efficiency of pattern matching in large data graphs (see chapter “Graph Indexing and Synopses”).

Notable examples include PathBlast (Kelley et al. 2004), SAGA (Tian et al. 2006), NetAlign (Liang et al. 2006), and IsoRank (Singh et al. 2008) that target bioinformatics applications. G-Ray (Tong et al. 2007) aims to maintain the shape of the query via effective path exploration. TALE (Tian and Patel 2008) uses edge misses to measure the quality of a match. NESS and NeMa (Khan et al. 2011, 2013) incorporate more flexibility in graph pattern matching by allowing an edge in the query graph to be matched with a path up to a certain number of hops in the target graph. Both methods identify the top- k matches. This is useful in searching knowledge graphs and social networks.

Making Big Graphs Small

When the query class is inherently expensive, it is usually nontrivial to reduce the complex-

ity. The second category of research aims to identify a small and relevant fraction G_Q of large-scale graph G that suffice to provide approximate matches for query Q . That is, the match $Q(G_Q)$ is same or close to its exact counterpart $Q(G)$. Several strategies have been developed for graph pattern matching under this principle.

Making graph query bounded. Bounded evaluability of graph pattern queries aims to develop algorithms with provable computation cost that is determined by the small fraction of graph. Notable examples include query preserving graph compression (Fan et al. 2012a) and querying with bounded resources (Fan et al. 2014b; Cao et al. 2015). (1) Given a query class \mathcal{Q} , query-preserving compression applies a fast once-for-all preprocessing to compress data graph G to a small, query-able graph G_Q , which guarantees that for any query instance from \mathcal{Q} , $Q(G) = Q(G_Q)$. Compression schemes have been developed for graph pattern queries defined by reachability and graph simulation. (2) Querying with bounded resource adds additional size bound to G_Q . The method in Cao et al. (2015) makes use of a class of access constraints to estimate and fetch the amount of data needed to evaluate graph queries. When no such constraints exist, another method (Fan et al. 2014b) dynamically induces G_Q on the fly to give approximate matches. Both guarantee to access G_Q with bounded size. (3) Graph views are also used to achieve bounded evaluability, where query processing only refers to a set of materialized views with bounded size (Fan et al. 2014a).

Incremental evaluation. When data graph G bear changes ΔG , bounded evaluability is achieved by applying *incremental pattern matching* (Fan et al. 2013). While ΔG is typically small in practice, G_Q is induced by combining ΔG and a small fraction of G (called “affected area”) that are necessarily visited to update $Q(G)$ to its counterparts in the updated graph. An algorithm guarantees bounded computation if it incurs cost determined by the neighbors of G_Q (up to a certain hop) and Q only, independent of

the size of G . Bounded incremental algorithms are identified for several graph pattern queries including reachability and graph simulation (Fan et al. 2013).

Parallelizing sequential computation. When data graph G is too large for a single processor, bounded evaluability can be achieved by distributing G to several processors. Each processor copes with a fragment G_Q of G with a manageable size and bounded evaluability, guaranteed by partition, load balancing, and other optimization techniques. Existing models include bulk (synchronous and asynchronous) vertex-centric computation and graph-centric model , which are designed to parallelize node and graph-level computations, respectively (see chapter “► Parallel Graph Processing”).

Several characterizations of such models have been proposed. Notable examples include *scale independence* (Armbrust et al. 2009), which ensures that a parallel system enforces strict invariants on the cost of query execution as data size grows, and *parallel scalability* (Fan et al. 2014c, 2017), which ensures that the system has polynomial speed-up over sequential computation as more number of processors are used. A notable example is the techniques that aim to automatically parallelize sequential graph computation without recasting (Fan et al. 2017). The declarative platform defines parallel computation with three core functions for any processor – to support local computation, incremental computation (upon receiving new messages), and assembling of partial answers from workers. Several parallel scalable algorithms are developed under this principle for graph pattern queries including reachability and graph simulation (Fan et al. 2012b, 2014c).

User-Friendly Pattern Matching

A third category of research aims to develop algorithms that can better infer the search intent of graph queries, especially when end users have no prior knowledge of underlying graph data. We review two approaches specialized for graph pattern matching.

Graph keyword search. Keyword search over graphs allows users to provide a list of keywords and returns subtrees/subgraphs as interpretable answers. Various ranking criteria also exist to find the top- k answers, e.g., sum of all edge weights in the resulting tree/graph, sum of all path weights from root to each keyword in the tree, maximum pairwise distance among nodes, etc. Notable examples include BANKS (Aditya et al. 2002) and BLINKS (He et al. 2007) (see chapter “► Graph Exploration and Search” for details). Several keyword searches find answers that are connected subgraphs containing all the query keywords. Examples include r -clique (Kargar and An 2011), where the nodes in an r -clique are separated by a distance at most r . Query refinement and reformulation techniques are studied to convert the keyword queries to structured queries such as SPARQL. Beyond keyword queries, Zheng et al. (2017) develop a more interactive way of answering natural language queries over graph data. For more details of graph queries, see chapters “► Graph Query Languages” and “► Graph Query Processing.”

Graph query by example. A relatively new paradigm for user-friendly graph querying is graph-query-by-example. Query-by-example (QBE) has a positive history in relational databases, HTML tables, and entity sets. Exemplar query (Mottin et al. 2014) and GQBE (Jayaram et al. 2015) adapted similar ideas over knowledge graphs. In particular, the user may find it difficult how to precisely write her query, but she might know a few answers to her query. A graph query-by-example system allows her to input the answer tuple as a query and returns other similar tuples that are present in the target graph. The underlying system follows a two-step approach. Given the input example tuple(s), it first identifies the query graph that captures the user’s query intent. Then, it evaluates the query graph to find other relevant answer tuples. Major techniques of graph query by example are introduced in chapter “► Graph Exploration and Search.”

Key Applications

Graph pattern matching has been widely adopted in accessing and understanding network data. Representative applications include community detection in social networks, attack detection for cyber security, querying Web and knowledge bases as knowledge graphs, pattern recognition in brain and healthcare, sensor network in smart environment, and Internet of Things.

Future Directions

The open challenges remain to be the designing of efficient and scalable graph pattern query models and algorithms to meet the need of querying big graphs that are heterogeneous, large, and dynamic. Another challenge is the lack of support for declarative languages for more flexible graph pattern queries beyond SPARQL. A third challenge is to enable fast and scalable pattern matching with quality guarantees in incomplete and uncertain graphs, with applications in data fusion and provenance of high-value graph data sources. These call for new scalable techniques for graph pattern query evaluation.

Cross-References

- Graph Data Models
- Graph Exploration and Search
- Graph Query Languages
- Graph Query Processing
- Parallel Graph Processing

References

- Aditya B, Bhalotia G, Chakrabarti S, Hulgeri A, Nakhe C, Parag P, Sudarshan S (2002) BANKS: browsing and keyword searching in relational databases. In: VLDB
 Armbrust M, Fox A, Patterson D, Lanham N, Trushkowsky B, Trutna J, Oh H (2009) Scads: scale-independent storage for social computing applications. arXiv preprint arXiv:09091775

- Cao Y, Fan W, Huai J, Huang R (2015) Making pattern queries bounded in big graphs. In: ICDE, pp 161–172
- Cordella LP, Foggia P, Sansone C, Vento M (2004) A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans Pattern Anal Mach Intell* 26(10):1367–1372
- Fan W, Li J, Ma S, Tang N, Wu Y, Wu Y (2010) Graph pattern matching: from intractable to polynomial time. *VLDB* 3(1–2):264–275
- Fan W, Li J, Ma S, Tang N, Wu Y (2011) Adding regular expressions to graph reachability and pattern queries. In: ICDE, pp 39–50
- Fan W, Li J, Wang X, Wu Y (2012a) Query preserving graph compression. In: SIGMOD, pp 157–168
- Fan W, Wang X, Wu Y (2012b) Performance guarantees for distributed reachability queries. *VLDB* 5(11):1304–1316
- Fan W, Wang X, Wu Y (2013) Incremental graph pattern matching. *TODS* 38(3):18:1–18:47
- Fan W, Wang X, Wu Y (2014a) Answering graph pattern queries using views. In: ICDE
- Fan W, Wang X, Wu Y (2014b) Querying big graphs within bounded resources. In: SIGMOD
- Fan W, Wang X, Wu Y, Deng D (2014c) Distributed graph simulation: impossibility and possibility. *VLDB* 7(12):1083–1094
- Fan W, Xu J, Wu Y, Yu W, Jiang J, Zheng Z, Zhang B, Cao Y, Tian C (2017) Parallelizing sequential graph computations. In: SIGMOD
- He H, Singh A (2008) Graphs-at-a-time: query language and access methods for graph databases. In: SIGMOD
- He H, Wang H, Yang J, Yu PS (2007) BLINKS: ranked keyword searches on graphs. In: SIGMOD
- Jayaram N, Khan A, Li C, Yan X, Elmasri R (2015) Querying knowledge graphs by example entity tuples. *TKDE* 27(10):2797–2811
- Kargar M, An A (2011) Keyword search in graphs: finding R-cliques. *VLDB* 4(10):681–692
- Kelley BP, Yuan B, Lewitter F, Sharan R, Stockwell BR, Ideker T (2004) PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Res* 32: 83–88
- Khan A, Ranu S (2017) Big-graphs: querying, mining, and beyond. In: Handbook of big data technologies. Springer, Cham, pp 531–582
- Khan A, Li N, Guan Z, Chakraborty S, Tao S (2011) Neighborhood based fast graph search in large networks. In: SIGMOD
- Khan A, Wu Y, Aggarwal C, Yan X (2013) NeMa: fast graph search with label similarity. *VLDB* 6(3): 181–192
- Lee J, Han WS, Kasperovics R, Lee JH (2012) An in-depth comparison of subgraph isomorphism algorithms in graph databases. *VLDB* 6(2):133–144
- Liang Z, Xu M, Teng M, Niu L (2006) NetAlign: a web-based tool for comparison of protein interaction networks. *Bioinformatics* 22(17):2175–2177
- Ma S, Cao Y, Fan W, Huai J, Wo T (2011) Capturing topology in graph pattern matching. *VLDB* 5(4):310–321
- Mottin D, Lissandrini M, Velegrakis Y, Palpanas T (2014) Exemplar queries: give me an example of what you need. *VLDB* 7(5):365–376
- Shang H, Zhang Y, Lin X, Yu J (2008) Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *VLDB* 1(1):364–375
- Singh R, Xu J, Berger B (2008) Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS* 105(35):12763–12768
- Tian Y, Patel JM (2008) TALE: a tool for approximate large graph matching. In: ICDE
- Tian Y, McEachin R, Santos C, States D, Patel J (2006) SAGA: a subgraph matching tool for biological graphs. *Bioinformatics* 23(2):232–239
- Tong H, Faloutsos C, Gallagher B, Eliassi-Rad T (2007) Fast best-effort pattern matching in large attributed graphs. In: KDD
- Ullmann JR (1976) An algorithm for subgraph isomorphism. *J ACM* 23:31–42
- Zhang S, Li S, Yang J (2009) GADDI: distance index based subgraph matching in biological networks. In: EDBT
- Zhao P, Han J (2010) On graph query optimization in large networks. In: VLDB
- Zheng W, Cheng H, Zou L, Yu JX, Zhao K (2017) Natural language question/answering: let users talk with the knowledge graph. In: CIKM

G

Graph Processing Frameworks

Arturo Diaz-Perez¹, Alberto Garcia-Robledo², and Jose-Luis Gonzalez-Compean¹

¹Cinvestav Tamaulipas, Victoria, Mexico

²Universidad Politecnica de Victoria, Victoria, Mexico

Synonyms

Graph parallel computation API; Graph processing library; Large-scale graph processing system

Definitions

A graph processing framework (GPF) is a set of tools oriented to process graphs. Graph vertices are used to model data and edges model

relationships between vertices. Typically, a GPF includes an input data stream, an execution model, and an application programming interface (API) having a set of functions implementing specific graph algorithms. In addition, some GPFs provide support for vertices and edges annotated with arbitrary properties of any kind and number.

Graphs are being used for modeling large phenomena and their relationships in different application domains such as social network analysis (SNA), biological network analysis, and link analysis for fraud/cybercrime detection. Since real graphs can be large, complex, and dynamic, GPFs have to deal with the three challenges of data growth: volume, velocity, and variety.

The programming API of GPFs is simple enough to facilitate efficient algorithm execution on graphs while abstracting away low-level details. On the one hand, the basic computation entity in a GPF can be graph-centric, vertex-centric, or edge-centric. On the other hand, as big data drives graph sizes beyond memory capacity of a single machine, GPFs partition and distribute data among computing nodes implementing a communication model.

Overview

There is a growing number of GPFs. Doekemeijer and Varbanescu (2014) review 83 different frameworks. To briefly explain the most relevant

aspects of GPFs, 11 proposals, shown in Table 1, have been selected in this entry. Four main dimensions can be used to classify GPF frameworks: (1) the target platform, (2) the computation model, (3) the graph processing approach, and (4) the communication model.

The target platform is one of the major distinctions between frameworks. Most of the GPFs are distributed memory frameworks (DMF), but there are some shared-memory frameworks (SMF) that exploit the parallel capabilities of a single machine.

Many graph processing algorithms need to traverse the graph in some way, and they are iterative until a global state is reached. Because of that, the computation model can be seen as a sequence of MapReduce invocations requiring to pass the entire state of the graph from one stage to the next. DMFs, like Surfer, Pegasus, and GBase, are based on MapReduce.

The bulk-synchronous process (BSP) model was created to simplify the design of software for parallel systems. BSP is a two-step process: perform task computing on local data and communicate the results. Both are performed iteratively and synchronously repeating the two step as many iterations needed. Each compute/communication iteration is called a superstep, with synchronization of the parallel tasks occurring at the superstep barriers.

DMFs, like Pregel, GraphX, Giraph, and GiraphX, and the hybrid framework, Totem, are based on the BSP model of computation.

Graph Processing Frameworks, Table 1 Frameworks for processing large-scale graphs

Framework	Year	Platform	Computation	Approach	Communication
Surfer	2010	Distributed memory	MapReduce	Graph-centric	Message passing
Pegasus	2009	Distributed memory	MapReduce	Matrix-centric	Dataflow
GBase	2011	Distributed memory	MapReduce	Matrix-centric	Dataflow
Pregel	2010	Distributed memory	BSP-based	Vertex-centric	Message passing
GraphX	2013	Distributed memory	BSP-based	Graph-centric	Dataflow
Giraph	2012	Distributed memory	BSP-based	Vertex-centric	Dataflow
GiraphX	2013	Distributed memory	BSP-based	Vertex-centric	Shared memory
Gelly	2015	Distributed memory	Delta iterative/GSA	Vertex-centric	Message passing
GraphLab	2010	Single machine/Distributed memory	GAS model	Vertex-centric	Shared memory
GraphChi	2012	Single machine	Sequential model	Vertex-centric, PSW	Shared memory
Totem	2012	Hybrid CPU + GPU	BSP-based	Graph-centric	Hybrid CPU + GPU

The Gelly framework supports two different alternative computation models: the delta iterative model and the Gather, Sum, and Apply (GSA) model. The delta model supports incremental iterations, in which only selected elements of the solution are updated and only the dependent elements are recomputed.

According to the GSA model, a vertex pulls information from its neighbors, contrary to the conventional vertex-centric approach where the information is pushed to the neighbors.

GraphLab is based in the gather, apply, and scatter (GAS) model, similar to BSP but having a synchronous or asynchronous execution mode. A different framework is GraphChi, a single-machine implementation framework relying on a parallel sliding window mechanism for processing large graphs from disk. Pegasus and GBase follow a matrix representations of graphs paying attention on the matrix computations and storage.

Regarding the graph processing approach, most frameworks employ a vertex-centric one, also known as the “think like a vertex” (TLAV) approach, in which user-defined programs center on a vertex rather than a graph. TLAV improves locality and scalability and provides a natural way to express many iterative graph algorithms. In contrast, frameworks like Surfer, GraphX, and Totem use a graph-centric approach performing computations on graph partitions.

GPFs can be classified also according to their memory model to share information. GiraphX, GraphLab, and GraphChi are based on a global shared-address memory space. Surfer and Pregel are elaborations of a pure message-passing model in which state is local and messages flow through the system to update external entities. Totem’s hybrid model is a combination of both shared-memory and message-passing models.

In the dataflow communication model, the flow of information goes from a vertex to its neighbors. In DMFs, the dataflow communication model is also to indicate that state flows through the system toward the next phase of computation.

The next sections are devoted to describe in more detail each of the GPFs listed in Table 1.

Review of Graph Processing Frameworks

MapReduce-Based Graph Processing Frameworks

Surfer is one of the first attempts to create a large-scale graph processing engine. It is based upon to basic primitives for programmers: MapReduce and propagation (Chen et al. 2010). MapReduce is used to process key-value pairs in parallel, and propagation is used to transfer information along edges.

Surfer employs a partition strategy dividing the large graph into many partitions of similar sizes. Each computer node in the distributed system holds several graph partitions and manages propagation doing as many local operations as it can before exchanging messages with other nodes in the system.

Pegasus is a large-scale graph mining library that has been implemented on top of the Hadoop framework (Kang et al. 2011b). Pegasus has built-in functions to perform typical graph mining tasks such as computing the diameter of a graph, performing a graph traversal, and finding the connected components. Pegasus uses the matrix-vector algebraic approach to perform most of its operations.

Pegasus has been utilized to discover patterns on near-cliques and triangles in large-scale graphs by MapReduce-based applications.

GBase is also a MapReduce-based framework focused in partitioning the graph into blocks and using block compression to store each block (Kang et al. 2011a). Vertex distribution is done by placing vertices belonging to same partition near to each other.

BSP Graph Processing Frameworks

Pregel system (Malewicz et al. 2010), introduced by Google, is the first BSP implementation providing an API specifically tailored for graph algorithms following the TLAV model. Each vertex executes the user-defined vertex function and then sends results to neighbors along graph edges. Pregel is a DMF where graph vertices are allocated in the different machines achieving that

all associated neighbors are assigned to the same node.

Pregel deals with the underlying organization of resource allocation of the graph data and splits computation into BSP-style supersteps. Supersteps always end with a synchronization barrier, which guarantees that messages sent in a given superstep are received at the beginning of the next superstep.

Vertices may change states between active and inactive, depending on the overall state of execution. Pregel terminates when all vertices halt and no more messages are exchanged.

GraphX is a distributed graph engine built on top of Spark, which is a MapReduce-like general data-parallel engine currently released as an Apache project (Xin et al. 2013). Spark’s programming model consists of:

1. An in-memory data abstraction called resilient distributed datasets (RDD), and
2. A set of deterministic parallel operations on RDDs that can be invoked using data primitives such as map, filter, join, and group by.

A Spark’s program is a directed acyclic graph (DAG) of Spark’s primitives, which are applied in a set of initial or transformed RDDs.

GraphX extends Spark’s RDD abstraction to introduce resilient distributed graph (RDG) which associates records with vertices and edges and provides a collection of expressive computational graph primitives.

GraphX relies on a flexible vertex-cut partitioning to encode graphs as horizontally partitioned collections. A vertex-cut partitioning promotes to evenly assign edges to machines leaving vertices to span multiple machines. A vertex-data table is used to represent each vertex partition; an edge table is used to represent each edge partition, i.e., those vertices having an edge crossing from one partition to another. A vertex map indexes all copies for each vertex.

Apache Flink is a stream processing framework developed by the Apache Software Foundation. Flink’s runtime provides support for the

execution of iterative algorithms natively. Gelly is Apache Flink’s graph processing API and library.

Gelly’s programming model exploits Flink’s delta iteration operators. Delta iterations enable Gelly to minimize the number of vertices that need to be recomputed during an iteration as the iterative algorithm. This can enhance performance when some vertices converge to their final value faster than others.

Gelly also supports the GSA model functions defined by the user and wrapped in map, reduce, and join operators. GSA also makes use of delta iteration operators.

Apache Giraph is an open-source project which uses Java to implement the Pregel specification, rather than C/C++, on the top of the Hadoop framework infrastructure (Avery 2011).

Giraph uses a vertex-centric programming abstraction adapting the BSP model. Giraph runs synchronous graph processing jobs as map-only jobs and uses Hadoop (HDFS) for storing graph data. Giraph employs Apache Zookeeper for periodic checkpointing, failure recovery, and coordinating superstep execution. Giraph is executed in-memory which can speed up job execution; however, it can fail processing a workload beyond the physical available memory.

Single-Machine Graph Processing Frameworks

An important issue of TLAV frameworks is how data is shared between vertex programs. In shared memory, vertex data are shared variables that can be directly read or modified by other vertex programs.

Shared memory is often implemented by TLAV frameworks developed for a single machine. Instead of dealing with consistency accessing remote vertices, using a mutual exclusion protocol, access to vertices stored in shared memory can be serialized.

GiraphX is a synchronous shared-memory implementation following the Giraph model (Tasci and Demirbas 2013). Serialization of shared vertex is provided through locking of border vertices, without local cached copies.

GiraphX showed a reduced overhead compared to message-passing in Giraph.

GraphLab is an open-source GPF implemented in C++, formerly started at CMU and currently supported by GraphLab Inc. GraphLab is a shared-memory abstraction using gather, apply, and scatter (GAS) model similar to BSP model employed in Pregel (Low et al. 2012) but with some differences. In the GAS model, a vertex collects information about its neighborhood in the gather phase, performs the computations in the apply phase, and in the scatter phase updates its adjacent vertices and edges.

GraphChi is a centralized system, implemented in C++, that can process massive graphs from secondary storage in a single machine (Kyrola et al. 2012). It uses a parallel sliding window (PSW) mechanism to process very large graphs from disk moving a fraction of the graph to memory and requiring a small number of sequential disk accesses.

GraphChi partitions the input graph into subgraphs where edges are sorted by the source id. A sequential stream is used to load the edges in memory. Using CSR and CSC graph representations, GraphChi avoids finding graph cuts which is a costly operation.

Modern single-node systems use both CPU and graphical processing units (GPUs) to accelerate the processing of large graphs. Host processing units are optimized to fast sequential processing, while GPUs bring massive hardware multithreading able to mask memory access latency.

Totem is the most representative hybrid framework combining GPU and CPU processing (Gharaibeh et al. 2013). Totem is also a BSP-based framework in which each processing element executes computations asynchronously on its local memory during the computation phase.

Totem's important contribution is that, for some real-world graphs, the communication overhead between CPUs and GPUs can be significantly reduced becoming negligible relative to the processing time.

Example of Application

The degree distribution is a useful tool to characterize the topology of real-world graphs. A power law degree distribution implies that a network has only a few vertices with very high degree, called hubs.

The following example illustrates the procedure to obtain the degree distribution of a large-scale social network by exploiting the API of a GPF in a distributed computing environment. The example shows how to obtain the degree distribution of the LiveJournal social network dataset, a graph with four million vertices and around 68 million links.

The degree distribution of a graph G can be defined as $P(k)$, the fraction of vertices in a graph that have degree k for all different k .

A general distributed computing strategy to calculate the degree distribution $P(k)$ with a GPF is as follows:

- Step 1: Use the GPF API to load and distribute the graph to all computing nodes.
- Step 2: Use the GPF API to calculate the total number of vertices n .
- Step 3: Obtain the list of all vertex degrees by letting each vertex to calculate its own degree in a distributed fashion with the help of the GPF API.
- Step 4: Calculate the degree histogram n_k , the frequency of each different degree in the list, by applying a distributed group by degree operation.
- Step 5: Calculate the degree distribution $P(k)$ by letting each degree frequency n_k to divide itself by the total number of vertices in the graph n , again in a distributed fashion.
- Step 6: Collect all the $P(k)$ data in a single computing node.

As the reader may notice, steps 1 through 3 are performed with the help of the graph processing abstractions provided by the GPF, whereas steps 4 through 6 can be expressed by using a conventional MapReduce approach. The following is the implementation of the previously described

steps with the Gelly framework API in the Scala language:

```

import org.apache.flink.api.scala._
import org.apache.flink.graph.scala._
import org.apache.flink.types.NullValue

object DegreeDistribution {
  def main(args: Array[String]) {

    // Set up the execution environment
    val env = ExecutionEnvironment.getExecutionEnvironment

    // Step 1: load and distribute the graph data
    val graph = Graph.fromCsvReader[Long, NullValue](
      NullValue]()
    pathEdges = "data/soc-LiveJournal1.csv",
    env = env)

    // Step 2: calculate the number of vertices
    val n = graph.numberOfVertices()

    // Step 3: calculate (vertex ID, degree) for all vertices
    val degrees = graph.getDegrees()

    // Steps 4 and 5: calculate the degree distribution
    val degreeDist = degrees
      .map { degreeTuple =>
        degreeTuple._2
      }
      .map { (_, 1) }
      .groupBy(0)
      .sum(1)
      .map { degreeCount =>
        (degreeCount._1, degreeCount._2 / n.toDouble)
      }

    // Step 6: collect and print out the result
    degreeDist.print()

  }
}

```

Step 1 is implemented with the `GraphCsvReader` method, which enables Gelly to load and build a Gelly graph dataset from an edge list CSV file of the LiveJournal graph, where the first field is the source vertex u , the second field is the target vertex v , and the third optional field is the edge value.

The `numberOfVertices` method is used to retrieve the number of vertices, implementing step 2. Likewise, the `getDegrees` method is used to compute the nodes degrees, implementing step 3. `getDegrees` builds a Flink dataset of vertex-degree pairs (u, k_u) for all vertices.

Steps 4–6 are implemented in a MapReduce style by using Flink's dataset methods to map, group, aggregate, and collect data. To implement step 4, the degree field of the (u, k_u) pairs dataset is separated to keep only the sequence of all vertex degrees $k_1, k_2 \dots k_n$. Then, each degree

k is paired with the value “1” through the `map` method to obtain a dataset of pairs $(k, 1)$. Next, the $(k, 1)$ pairs dataset is grouped by the degree field, summing up the value field to compute the degree frequencies with the `groupBy` and `sum` methods. This produces a new dataset of degree frequency pairs (k, n_k) .

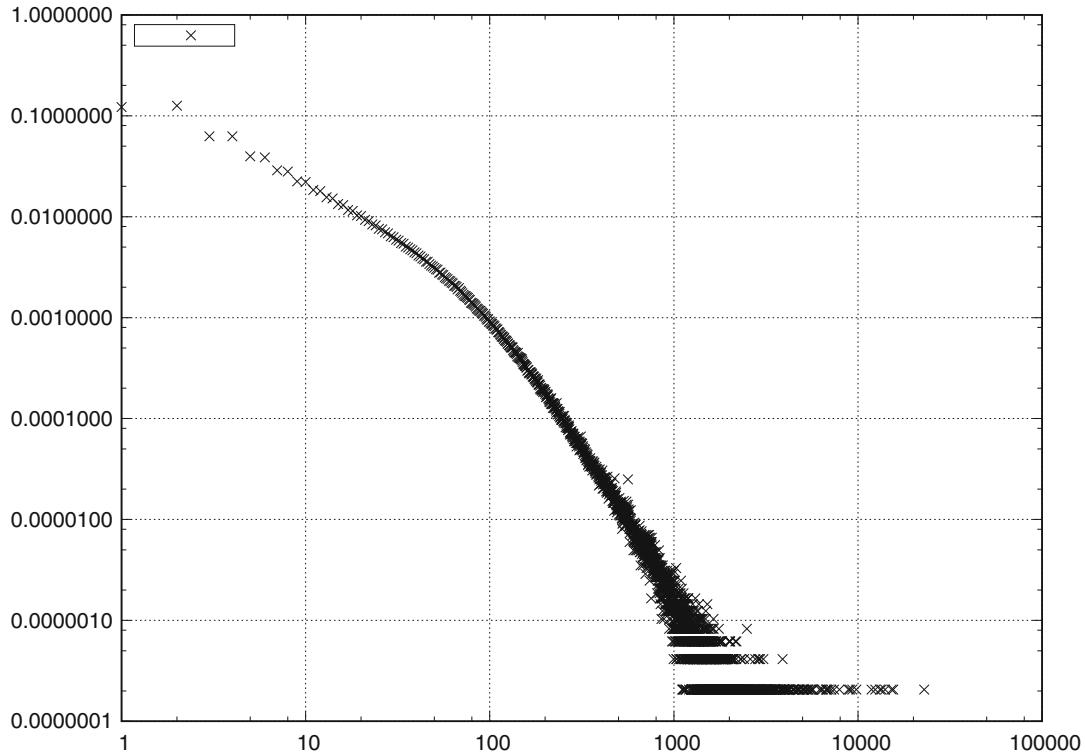
As for step 5, the frequency field of the (k, n_k) pair dataset is divided by the total number of vertices n , again through the `map` method, to obtain the final dataset of degree-probability pairs $(k, P(k))$, which represents the graph degree distribution. Step 6 collects and prints out the $(k, P(k))$ pairs dataset by making use of the `print` dataset method.

Figure 1 shows the log-log plot of the LiveJournal social network degree distribution obtained with the Gelly example. The close to linear plot shows that the graph degree distribution might follow a power law suggesting that the LiveJournal graph topology encodes relevant properties of the underlying social phenomenon.

Figure 2 shows the results of an experimental evaluation of the degree distribution Gelly application. All experiments were executed in a server including 32 cores (Intel Xeon 2.5 GHz), 64 RAM in which Centos 7, Java8, Scala 2.1.8, SBT 1.0.4, and Flink 1.4.0 were installed. Vertical axis shows the speedup and millions of TEPS (traversed edges per second) achieved by Gelly when using different numbers of threads (horizontal axis). The performance of Gelly is nonlinear improved in proportion to the number of threads. The speedup values for 2, 8, 16, and 32 cores were 1.93, 5.86, 7.39, and 7.58, respectively.

Future Directions for Research

Because of the increasing number and variety of available GPFs, users now face the challenge of selecting an appropriate platform for their specific application and requirements. First, a user might search for efficient implementations of graph algorithms. However, there exists a large variety of graph processing algorithms which can



Graph Processing Frameworks, Fig. 1 Plot of the LiveJournal social network degree distribution obtained with the Gelly GPF. The horizontal axis represents the degree

k . The vertical axis represents the degree distribution $P(k)$. Both axes are in logarithmic scale

be categorized into several groups by functionality and consumption of resources.

Efficient low-level implementation of such algorithms is subject to substantial implementation effort. Parallelizing graph algorithms with efficient performance is a very challenging task.

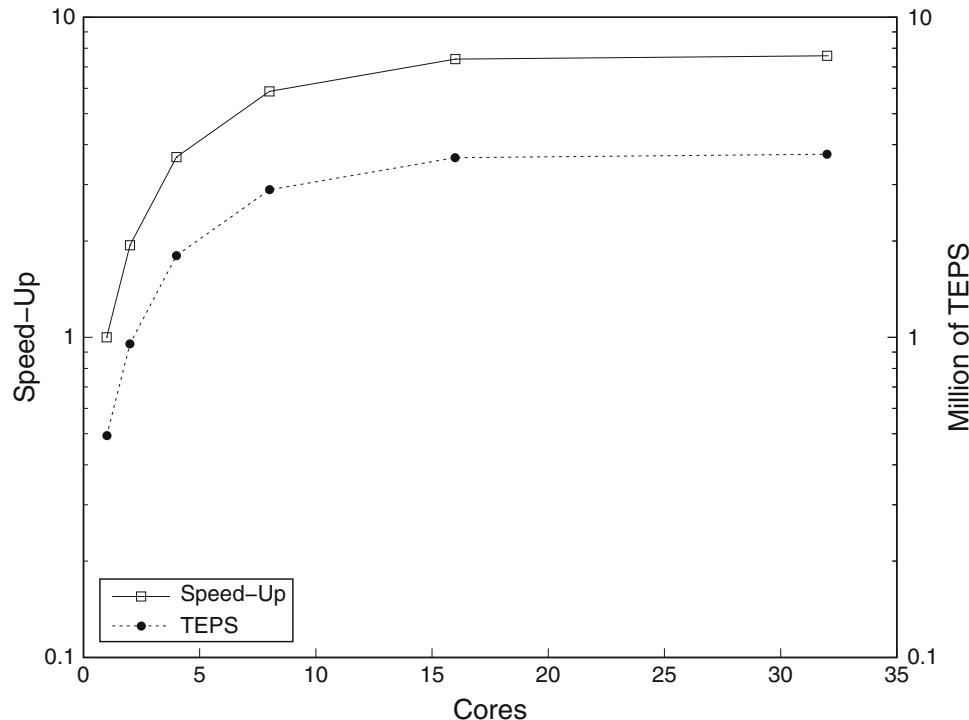
Although TLAV is a simple and easy to use approach to build graph processing applications, their implementation often leads to a significant amount of overhead. Efficient computing of graph partitions remains as a relevant topic in GPFs.

GPF performance is also a key aspect. Identifying the systems parameters to be monitored and the metrics that can be used to characterize GPF performance might be a difficult task. The general performance aspects can be raw processing power, resource usage, scalability, and processing overheads. These aspects can be observed by monitoring traditional system parameters like

lifetime of each processing job, the CPU and network load, the OS memory consumption, and the disk I/O, to name some parameters. Those observed parameters should be translated to relevant performance metrics such as vertices per second, edges per second, horizontal and vertical scalability, computation and communication time, and time overhead.

Datasets need to be considered also. In general, designing a good benchmark is challenging task due to the many aspects that should be considered which can influence the adoption and the usage scenarios of the benchmark. Unfortunately, there is clear lack of standard benchmarks that can be employed in GPF evaluation.

Finally, building GPFs to hybrid architectures is still under development. The main challenge in heterogeneous computing frameworks is to partition the graph in such a way that resulting partitions are mapped to processing elements



Graph Processing Frameworks, Fig. 2 Speed-up and TEPS produced by Gelly by using different numbers of threads of the LiveJournal Social Network degree distribution

according to their different architectural characteristics. In heterogeneous computing the power of different processors varies, so the load assigned to each processor should not be the same. Graph partitioning for hybrid architectures should focus on minimizing computation time rather than minimizing communications.

Cross-References

- ▶ [BSP Programming Model](#)
- ▶ [Distributed File Systems](#)
- ▶ [Large-Scale Graph Processing System](#)
- ▶ [Parallel Graph Processing](#)

References

- Avery C (2011) Giraph: large-scale graph processing infrastructure on hadoop. Proc Hadoop Summit Santa Clara 11(3):5–9
- Chen R, Weng X, He B, Yang M (2010) Large graph processing in the cloud. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 1123–1126
- Doekemeijer N, Varbanescu AL (2014) A survey of parallel graph processing frameworks. Delft University of Technology
- Gharaibeh ENA, Costa LB, Ripeanu M (2013) Totem: accelerating graph processing on hybrid CPU+GPU systems. In: GPU technology conference
- Kang U, Tong H, Sun J, Lin CY, Faloutsos C (2011a) Gbase: a scalable and general graph management system. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1091–1099
- Kang U, Tsourakakis CE, Faloutsos C (2011b) Pegasus: mining peta-scale graphs. Knowl Inf Syst 27(2): 303–325
- Kyrola A, Blelloch GE, Guestrin C (2012) Graphchi: large-scale graph computation on just a PC. In: USENIX symposium on OSDI
- Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM (2012) Distributed graphlab: a framework for machine learning and data mining in the cloud. Proc VLDB Endow 5(8):716–727
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system

- for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 135–146
- Tasci S, Demirbas M (2013) Giraphx: parallel yet serializable large-scale graph processing. In: European conference on parallel processing. Springer, pp 458–469
- Xin RS, Gonzalez JE, Franklin MJ, Stoica I (2013) Graphx: a resilient distributed graph system on spark. In: First international workshop on graph data management experiences and systems. ACM, p 2

ture types of a data model, in any combination desired.

In the context of graph data management, a *graph query language* (GQL) defines the way to retrieve or extract data which have been modeled as a graph and whose structure is defined by a graph data model. Therefore, a GQL is designed to support specific graph operations, such as graph pattern matching and shortest path finding.

G

Graph Processing Library

► Graph Processing Frameworks

Graph Properties

► Graph Invariants

Graph Query Languages

Renzo Angles¹, Juan Reutter², and Hannes Voigt³

¹Universidad de Talca, Talca, Chile

²Pontificia Universidad Católica de Chile, Santiago, Chile

³Dresden Database Systems Group, Technische Universität Dresden, Dresden, Germany

Overview

Research on graph query languages has at least 30 years of history. Several GQLs were proposed during the 1980s, most of them oriented to study and define the theoretical foundations of the area. During the 1990s, the research on GQLs was overshadowed by the appearance of XML, which was arguably seen as the main alternative to relational databases for scenarios involving semi-structured data. With the beginning of the new millennium, however, the area of GQLs gained new momentum, driven by the emergence of applications where cyclical relationships naturally occur, including social networks, the Semantic Web, and so forth. Currently GQLs are a fundamental tool that supports the manipulation and analysis of complex big data graphs. Detailed survey of graph query language research can be found in Angles et al. (2017b).

Key Research Findings

Most of the languages that have been proposed are based on the idea of matching a *graph pattern*. In designing a GQL one typically starts with graph patterns, to which several other features are added. Examples of these features include set operations, relational algebra operations, the addition of path queries, closing graph patterns through recursion, etc. In the following we describe how graph pattern matching works, as well as the most typical features commonly added to graph patterns.

Definitions

A query language is a high-level computer language for the retrieval and modification of data held in databases or files. Query languages usually consist of a collection of operators which can be applied to any valid instances of the data struc-

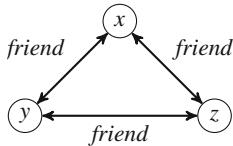
This work was supported by Iniciativa Científica Milenio Grant 120004.

Since our focus is on the fundamentals of GQLs, we concentrate on the simplest data model for graphs, i.e., directed labeled graphs. Later on, we review how these fundamentals are translated into practical query languages in more complex graph data models.

Let Σ be a finite alphabet. A *graph database* G over Σ is a pair (V, E) , where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of edges. That is, we view each edge as a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a -labeled edge from v to v' in G .

Graph Pattern Matching

The idea of patterns is to define small subgraphs of interest to look for in an input graph. For example, in a social network one can match the following pattern to look for a clique of three individuals that are all friends with each other:



Let $V = \{x, y, z, \dots\}$ be an infinite set of variables. In its simplest form, a graph pattern is simply a graph $P = (V_P, E_P)$ where all the nodes in V_P are replaced by variables. For example, the pattern above can be understood as a graph over alphabet $\Sigma = \{\text{friend}\}$ whose set of nodes is $\{x, y, z\}$ and whose edges are $\{(x, \text{friend}, y), (y, \text{friend}, x), (x, \text{friend}, z), (z, \text{friend}, x), (y, \text{friend}, z), (z, \text{friend}, y)\}$.

The semantics of patterns is given using the notion of a match. A match of a pattern $P = (V_P, E_P)$ over a graph $G = (V_G, E_G)$ is a mapping μ from variables to constants, such that the graph resulting from replacing each variable for the corresponding constant in the pattern is actually a subgraph of G , i.e., for every pattern node $x \in V_P$, we have that $\mu(x) \in V_G$, and for each edge $(x, a, y) \in E_P$, we have that $(\mu(x), a, \mu(y))$ belongs to E_G .

The problem of pattern matching has been extensively studied in the literature, and several other notions of matching have been considered. The basic notion defined above is known as a

homomorphism and constitutes the most common semantics. Additionally, one could also consider *simple path semantics*, which does not permit repeating nodes and used in, e.g., the languages G (Cruz et al. 1987b) and $G+$ (Cruz et al. 1989), *injective homomorphism*, that disallow variables to be mapped to the same element or semantics based on graph simulations (see, e.g., Miller et al. 2015 for a survey).

Although the set of matchings from P to G can be already seen as the set of answers for the evaluation of P over G , one is normally interested in tuples of nodes: if P uses variables x_1, \dots, x_n , then we define the result of evaluating P over G , which we denote by $P(G)$, as the set of tuples $\{(\mu(x_1), \dots, \mu(x_n)) \mid \mu \text{ is a match from } P \text{ to } G\}$.

Extensions. Graph patterns can be extended by adding numerous features. For starters, one could allow variables in the edge labels of patterns or allow a mixture between constant and variables in nodes or edges. In this case, mappings must be the identity on constant values, and if an edge (x, y, z) occurs in a pattern, then $\mu(y)$ must be a label in Σ , so that $(\mu(x), \mu(y), \mu(z))$ is an edge in the graph upon we are matching. Patterns can also be augmented with *projection*, and since the answer is a set of tuples, we can also define set operations (*union*, *intersection*, *difference*, etc.) over them. Likewise, one can define *filters* that force the matches to satisfy certain Boolean conditions such as $x \neq y$. Another operator is the left outer join, also known as the *optional* operator. This operator allows to query incomplete information (an important characteristic of graph databases).

Graph Patterns as Relational Database Queries.

Graph patterns can also be defined as *conjunctive queries* over the relational representation of graph databases (see, e.g., Abiteboul et al. 1995 for a good introduction on relational database theory). In order to do this, given an alphabet Σ , we define $\sigma(\Sigma)$ as the relational schema that consists of one binary predicate symbol E_a , for each symbol $a \in \Sigma$. For readability purposes, we identify E_a with a ,

for each symbol $a \in \Sigma$. Each graph database $G = (V, E)$ over Σ can be represented as a relational instance $\mathbf{D}(G)$ over the schema $\sigma(\Sigma)$: The database $\mathbf{D}(G)$ consists of all facts of the form $E_a(v, v')$ such that (v, a, v') is an edge in G (for this we assume that \mathbf{D} includes all the nodes in V). It is then not difficult to see that graph patterns are actually *conjunctive queries* over the relational representation of graphs, that is, formulas of the form $Q(\bar{x}) = \exists \bar{y} \phi(\bar{x}, \bar{y})$, where \bar{x} and \bar{y} are tuples of variables and $\phi(\bar{x}, \bar{y})$ is a conjunction of relational atoms from σ that use variables from \bar{x} to \bar{y} . For example, the pattern shown above can be written as the relational query $Q(x, y, z) = \text{friend}(x, y) \wedge \text{friend}(y, x) \wedge \text{friend}(x, z) \wedge \text{friend}(z, x) \wedge \text{friend}(y, z) \wedge \text{friend}(z, y)$.

The correspondence between patterns and conjunctive queries allows us to reuse the broad corpus on literature on conjunctive queries for our graph scenario. We also use this correspondence when defining more complex classes of graph patterns.

Path Queries

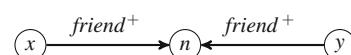
The idea of a path query is to select pairs of nodes in a graph whose relationship is given by a path (of arbitrary length) satisfying certain properties. For example, one could use a path query in a social network to extract all nodes that can be reached from an individual via friend-of-a-friend paths of arbitrary length.

The most simple navigational querying mechanism for graph databases is a *regular path query (RPQ)* (Abiteboul et al. 1999; Cruz et al. 1987a; Calvanese et al. 2002), which allows to specify navigation by means of regular expressions. Formally, an RPQ Q over Σ is a regular language $L \subseteq \Sigma^*$, and it is specified using a regular expression R . Their semantics is defined in terms of paths. A *path* ρ from v_0 to v_m in a graph $G = (V, E)$ is a sequence $(v_0, a_0, v_1), (v_1, a_1, v_2), \dots, (v_{m-1}, a_{m-1}, v_m)$, for some $m \geq 0$, where each (v_i, a_i, v_{i+1}) , for $i < m$, is an edge in E . In particular, all the v_i 's are nodes in V and all the a_j 's are letters in Σ . The *label* of ρ is the word $a_0 \dots a_{m-1} \in \Sigma^*$.

The idea behind regular path queries is to select all pairs of nodes whose labels belong to the language of the defining regular expression. That is, given a graph database $G = (V, E)$ and an RPQ R , the evaluation of R over G , denoted $\llbracket R \rrbracket_G$, is the set of all pairs $(v, v') \in V$ such that there is path ρ between v and v' and the label of ρ is a word in the language of R .

More Complex Path Queries. RPQs have been extended in a myriad of ways. For example, we have *two-way regular path queries* (2RPQs) (Calvanese et al. 2000) that add to RPQs the ability to traverse edge backward via inverse symbols a^- , *nested* regular path queries (Barceló et al. 2012b) that further extend 2RPQs with node tests similar to what is done with propositional dynamic logic or XPath expressions, and several proposals adding variables to path queries (Wood 1990; Santini 2012; Fionda et al. 2015). There are also proposals of more expressive forms of path queries, using, for example, context-free languages (Hellings 2014), and several formalisms aimed at comparing pieces of data that may be associated with the nodes in a graph (see, e.g., Libkin and Vrgoč 2012). Another type of path queries, called *property paths*, extends RPQs with a mild form of negation (Kostylev et al. 2015).

Adding Path Queries to Patterns. Path queries and patterns can be naturally combined to form what is known as *conjunctive regular path queries*, or CRPQs (Florescu et al. 1998; Calvanese et al. 2000). Just as with graph patterns and conjunctive queries, we can define CRPQs in two ways. From a pattern perspective, CRPQs amount to replacing some of the edges in a graph pattern by a regular expression, such as the following CRPQ, which looks for two paths with an indirect friend n in common:



However, a CRPQ over an alphabet Σ can also be seen as a conjunctive query over a relational representation of graphs that includes a predicate for each regular expression constructed from Σ .

In our case, the corresponding conjunctive query for the pattern above is written as $Q(x, y) = \text{friend}^+(x, n) \wedge \text{friend}^+(y, n)$. Note that here we are using n as a constant; since CRPQs are again patterns, they can be augmented with any of the features we listed for patterns, including constants, projection, set operations, filters, and outer joins. We can also obtain more expressive classes of queries by including more expressive forms of path queries: using 2RPQs instead of RPQs gives rise to C2RPQs, or conjunctive 2RPQs, and using nested regular expressions gives rise to CNREs, which are also known as *conjunctive nested path queries*, or CNPQs (Bienvenu et al. 2014; Bourhis et al. 2014). Finally, Barceló et al. (2012a) consider restricting several path queries in a CRPQ at the same time by means of regular relations. The resulting language is capable of expressing, for example, that the two paths of friends in the pattern above must be of the same length.

Algorithmic Issues. The most important algorithmic problem studied for GQLs is *query answering*. This problem asks, given a query Q with k variables, a graph G and a tuple \bar{a} of k values from G , whether \bar{a} belongs to the evaluation of Q over G . All of the classes of path queries we have discussed can be evaluated quite efficiently. For example, all of RPQs, 2RPQs, and NREs can be evaluated in linear time in the size of Q and of G . Barceló (2013) provides a good survey on the subject.

When adding path queries to patterns, one again hopes that the resulting language will not be more difficult to evaluate than standard patterns or conjunctive queries. And indeed this happens to be the case: The evaluation problem for CRPQs, C2RPQs, and CNREs is NP-complete just as patterns. In fact, one can show this for any pattern augmented with a form of path query, as long as the evaluation for these path queries is in PTIME.

The second problem that has received considerable attention is *query containment*. This problem asks, given queries Q_1 and Q_2 , whether the answers of Q_1 are always contained in the

answers of Q_2 . Since all queries we presented are based on graph patterns, the lower bound for this problem is NP, as containment is known to be NP-complete already for simple graph patterns. In contrast to evaluation, adding path queries to patterns can significantly alter the complexity of containment: for CRPQs the problem is already EXPSPACE-complete (Calvanese et al. 2000), and depending on the features considered, it may even become undecidable.

Beyond Patterns

For the navigational languages, we have seen, thus far, paths are the only form of recursion allowed, but to express certain types of queries, we may require more expressive forms of recursion. As an example, suppose now our social network admits labels *friends* and *works_with*, the latter for joining two nodes that are colleagues. Now imagine that we need to find all nodes x and y that are connected by a path where each subsequent pair of nodes in the path is connected by both *friend* and *works_with* relations. We cannot express this query as a regular expression between paths, so instead what we do is to adopt *Datalog* as a framework for expressing the repetitions of patterns themselves. Consider, for example, a rule of the form:

$$P(x, y) \leftarrow \text{friend}(x, y), \text{works_with}(x, y).$$

This rule is just another way of expressing the pattern that computes all pairs of nodes connected both by *friend* and *works_with* relations. In this case, P represents a new relation, known as an *intensional predicate*, that is intended to compute all pairs of nodes that are satisfied by this pattern. We can then use P to specify the rule:

$$\text{Ans}(x, y) \leftarrow P^+(x, y).$$

Now both rules together form what is known as a *regular query* (Reutter et al. 2015) or nested positive 2RPQs (Bourhis et al. 2014, 2015). The idea is that $P^+(x, y)$ represents all pairs x, y connected by a path of nodes, all of them satisfying the pattern P (i.e., connected both by *friend* and *works_with* relations). Regular queries can be understood as the language resulting of extending

graph patterns with edges that can be labeled not only with a path query but with any arbitrary repetition of a binary pattern.

The idea of using Datalog as a graph query language for graph databases comes from Consens and Mendelzon (1990), where it was introduced under the name GraphLog. GraphLog opened up several possibilities for designing new query languages for graphs, but one of the problems of GraphLog was that it was too expressive, and therefore problems such as query containment were undecidable for arbitrary GraphLog queries. However, interest in Datalog for graphs has returned in the past few years, as researchers started to restrict Datalog programs in order to obtain query languages that can allow expressing the recursion or repetition of a given pattern, but at the same time maintaining good algorithmic properties for the query answering problem. For example, the complexity for evaluating regular queries is the same as for C2RPQs, and the containment problem is just one exponential higher. Besides regular queries, other noteworthy graph languages follow by restricting to monadic Datalog programs (Rudolph and Krötzsch 2013), programs whose rules are either guarded or frontier-guarded (Bourhis et al. 2015; Bienvenu et al. 2015), and first-order logic with transitive closure (Bourhis et al. 2014).

Composability

A particularly important feature of query languages is *composability*. Composable query languages allow to formulate queries with respect to results of other queries expressed in the same language. Composability is an important principle for the design of simple but powerful programming languages in general and query languages in particular (Date 1984). To be composable, the input and the output of a query must have the same (or a compatible) data model.

CQs (and C2RPQs) are not inherently composable, since their output is a relation and their input is a graph. In contrast, RPQs (and 2RPQs) allow query composition, since the set of pairs of nodes resulting from an RPQ can be interpreted as edges obviously. Regular queries

are composable for the same reason. In general, Datalog-based graph query languages are composable over relations but not necessarily over graphs.

In the context of modern data analytics, data is collected to a large extent automatically by hardware and software sensors in fine granularity and low abstraction. Where users interact with data, they typically think, reason, and talk about entities of larger granularity and higher abstraction. For instance, social network data is collected in terms of *friend* relationship and individual messages sent from one person to another, while the user is often interested in communities, discussion threads, topics, etc. User-level concepts are often multiple abstraction levels higher than the concepts in which data is captured and stored. The query language of database systems is the main means for users to derive information in terms of their user-level concepts for a database oblivious of user's concepts. To offer a capability of conceptual abstraction, a query language has to (1) be able to create entirely new entities within the data model to lift data into higher-level concepts and (2) be composable over its data model, so that the users can express a stack of multiple such conceptual abstraction steps with the same language.

RPQs (and 2RPQs) and regular queries allow conceptual abstraction for edges. For instance, *ancestor* edges can be derived from sequences of *parent* edges. Conceptual abstraction for nodes has been considered only recently (Rudolf et al. 2014; Voigt 2017) and allows to derive (or construct) an entirely new output graph from the input graph. In the simplest form of *graph construction*, a new node is derived from each tuple resulting from a pattern match (C2RPQs, etc.), e.g., deriving a *marriage* node from every pair of persons that are mutually connected by *married – to* edges. The more general and more expressive variant of graph construction includes aggregation and allows to derive a new node from every group of tuples given a set of variables as grouping keys, e.g., deriving a *parents* node from every group of adults all connected by *parent – to* the same child.

Key Applications

There is a vast number of GQLs that have been proposed and implemented by graph databases and an even larger number of GQLs that are used in specific application domains (see, e.g., Dries et al. 2009; Brijder et al. 2013; Martín et al. 2011). In the following we focus on GQLs designed for the two most popular graph data models: property graphs and RDF.

Property Graph Databases

A property graph is a directed labeled multigraph with the special characteristic that each node or edge can maintain a set (possibly empty) of properties, usually represented as key-value pairs. The property graph data model is very popular in current graph database systems, including Neo4j, Oracle PGX, Titan, and Amazon Neptune.

There is no standard query language for property graphs although some proposals are available. Blueprints (<http://tinkerpop.apache.org/>) was one of the first libraries created for manipulating property graphs. Blueprints is analogous to the JDBC, but for graph databases. Gremlin (Rodriguez 2015) is a graph-based programming language for property graphs developed within the TinkerPop open-source project. Gremlin makes extensive use of XPath to support complex graph traversals. Cypher is a declarative query language defined for the Neo4j graph database engine, originally, and by now adopted by other implementers. Cypher supports basic graph patterns and basic types of regular path queries (i.e., paths with a fixed edge label or paths with any labels) with a no-repeating-edges matching semantics. The development of Cypher is a still ongoing community effort (<http://www.opencypher.org/>).

PGQL (van Rest et al. 2016) is a query language for property graphs recently developed as part of Oracle PGX, an in-memory graph analytic framework. PGQL defines a SQL-like syntax that allows to express graph pattern matching queries, regular path queries (with conditions on labels and properties), graph construction, and query composition.

G-CORE (Angles et al. 2017a) is a language recently proposed by the Linked Data Benchmark Council (LDBC). G-CORE advertises regular queries, nested weighted shortest path queries, graph construction with aggregation, and composability as the core feature of future graph query languages on property graphs.

RDF Databases

The Resource Description Framework (RDF) is a W3C recommendation that defines a graph-based data model for describing and publishing data on the web. This data model gains popularity in the context of managing web data, giving place to the development of RDF databases (also called triplestores). Along with these database systems, several RDF query languages were developed (we refer the reader to a brief survey by Haase et al. 2004). Currently, SPARQL (Prud'hommeaux and Seaborne 2008) is the standard query language for RDF. SPARQL was designed to support several types of complex graph patterns and, in its latest version, SPARQL 1.1 (Harris and Seaborne 2013), adds support for negation, regular path queries (called *property paths*), subqueries, and aggregate operators. The path queries support reachability tests.

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Processing](#)

References

- Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley, Reading
- Abiteboul S, Buneman P, Suciu D (1999) Data on the Web: from relations to semistructured data and XML. Morgan Kauffman, San Francisco
- Angles R, Arenas M, Barceló P, Boncz PA, Fletcher GHL, Gutierrez C, Lindaaker T, Paradies M, Plantikow S, Sequeda J, van Rest O, Voigt H (2017a) G-CORE: a core for future graph query languages. The computing research repository abs/1712.01550

- Angles R, Arenas M, Barceló P, Hogan A, Reutter JL, Vrgoc D (2017b) Foundations of modern query languages for graph databases. *ACM Comput Surv* 68(5):1–40
- Barceló P (2013) Querying graph databases. In: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2013, pp 175–188
- Barceló P, Libkin L, Lin AW, Wood PT (2012a) Expressive languages for path queries over graph-structured data. *ACM Trans Database Syst (TODS)* 37(4):31
- Barceló P, Pérez J, Reutter JL (2012b) Relative expressiveness of nested regular expressions. In: Proceedings of the Alberto Mendelzon workshop on foundations of data management (AMW), pp 180–195
- Bienvenu M, Calvanese D, Ortiz M, Simkus M (2014) Nested regular path queries in description logics. In: Proceeding of the international conference on principles of knowledge representation and reasoning (KR)
- Bienvenu M, Ortiz M, Simkus M (2015) Navigational queries based on frontier-guarded datalog: preliminary results. In: Proceeding of the Alberto Mendelzon workshop on foundations of data management (AMW), p 162
- Bourhis P, Krötzsch M, Rudolph S (2014) How to best nest regular path queries. In: Informal Proceedings of the 27th International Workshop on Description Logics
- Bourhis P, Krötzsch M, Rudolph S (2015) Reasonable highly expressive query languages. In: Proceeding of the international joint conference on artificial intelligence (IJCAI), pp 2826–2832
- Brijder R, Gillis JJM, Van den Bussche J (2013) The DNA query language DNAQL. In: Proceeding of the international conference on database theory (ICDT). ACM, pp 1–9
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2000) Containment of conjunctive regular path queries with inverse. In: Proceeding of the international conference on principles of knowledge representation and reasoning (KR), pp 176–185
- Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2002) Rewriting of regular expressions and regular path queries. *J Comput Syst Sci (JCSS)* 64(3):443–465
- Consens M, Mendelzon A (1990) Graphlog: a visual formalism for real life recursion. In: Proceeding of the ACM symposium on principles of database systems (PODS), pp 404–416
- Cruz I, Mendelzon A, Wood P (1987a) A graphical query language supporting recursion. In: ACM special interest group on management of data 1987 annual conference (SIGMOD), pp 323–330
- Cruz IF, Mendelzon AO, Wood PT (1987b) A graphical query language supporting recursion. In: Proceeding of the ACM international conference on management of data (SIGMOD), pp 323–330
- Cruz IF, Mendelzon AO, Wood PT (1989) G+: recursive queries without recursion. In: Proceeding of the international conference on expert database systems (EDS). Addison-Wesley, pp 645–666
- Date CJ (1984) Some principles of good language design (with especial reference to the design of database languages). *SIGMOD Rec* 14(3):1–7
- Dries A, Nijssen S, De Raedt L (2009) A query language for analyzing networks. In: Proceeding of the ACM international conference on information and knowledge management (CIKM). ACM, pp 485–494
- Fionda V, Pirrò G, Consens MP (2015) Extended property paths: writing more SPARQL queries in a succinct way. In: Proceeding of the conference on artificial intelligence (AAAI)
- Florescu D, Levy AY, Suciu D (1998) Query containment for conjunctive queries with regular expressions. In: Proceeding of the ACM symposium on principles of database systems (PODS), pp 139–148
- Haase P, Broekstra J, Eberhart A, Volz R (2004) A comparison of RDF query languages. In: Proceeding of the international Semantic Web conference (ISWC), pp 502–517
- Harris S, Seaborne A (2013) SPARQL 1.1 query language. W3C recommendation. <http://www.w3.org/TR/sparql11-query/>
- Hellings J (2014) Conjunctive context-free path queries. In: Proceeding of the international conference on database theory (ICDT), pp 119–130
- Kostylev EV, Reutter JL, Romero M, Vrgoč D (2015) SPARQL with property paths. In: Proceeding of the international Semantic Web conference (ISWC), pp 3–18
- Libkin L, Vrgoč D (2012) Regular path queries on graphs with data. In: Proceeding of the international conference on database theory (ICDT), pp 74–85
- Martín MS, Gutierrez C, Wood PT (2011) SNQL: a social networks query and transformation language. In: Proceeding of the Alberto Mendelzon workshop on foundations of data management (AMW)
- Miller JA, Ramaswamy L, Kochut KJ, Fard A (2015) Research directions for big data graph analytics. In: Proceeding of the IEEE international congress on big data, pp 785–794
- Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. <http://www.w3.org/TR/rdf-sparql-query/>
- Reutter JL, Romero M, Vardi MY (2015) Regular queries on graph databases. In: Proceeding of the international conference on database theory (ICDT), pp 177–194
- Rodriguez MA (2015) The Gremlin graph traversal machine and language. In: Proceeding of the international workshop on database programming languages. ACM
- Rudolf M, Voigt H, Bornhövd C, Lehner W (2014) SynopSys: foundations for multidimensional graph analytics. In: Castellanos M, Dayal U, Pedersen TB, Tatbul N (eds) BIRTE'14, business intelligence for the real-time enterprise, 1 Sept 2014. Springer, Hangzhou, pp 159–166
- Rudolph S, Krötzsch M (2013) Flag & check: data access with monadically defined queries. In: Proceeding of the symposium on principles of database systems (PODS). ACM, pp 151–162

- Santini S (2012) Regular languages with variables on graphs. *Inf Comput* 211:1–28
- van Rest O, Hong S, Kim J, Meng X, Chafi H (2016) PGQL: a property graph query language. In: Proceeding of the workshop on graph data-management experiences and systems (GRADES)
- Voigt H (2017) Declarative multidimensional graph queries. In: Proceeding of the 6th European business intelligence summer schoole (BISS). LNBP, vol 280. Springer, pp 1–37
- Wood PT (1990) Factoring augmented regular chain programs. In: Proceeding of the international conference on very large data bases (VLDB), pp 255–263

Graph Query Processing

S. Salihoglu¹ and N. Yakovets²

¹University of Waterloo, Waterloo, ON, Canada

²Eindhoven University of Technology,
Eindhoven, Netherlands

Definitions

While being eminently useful in a wide variety of application domains, the high expressiveness of graph queries makes them hard to optimize and, hence, challenging to process efficiently. We discuss a number of state-of-the-art approaches which aim to overcome these challenges, focusing specifically on planning, optimization, and execution of two commonly used types of declarative graph queries: subgraph queries and regular path queries.

Overview

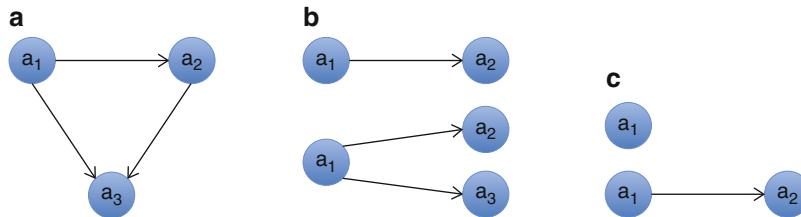
In this chapter, we give a broad overview of the general techniques for processing two classes of graph queries: *subgraph queries*, which find instances of a *query subgraph* in a larger input graph, and *regular path queries* (RPQs), which find pairs of nodes in the graph connected by a path matching a given regular expression. Certainly, software that manage and process graph-structured data, such as RDF

triple stores (Neumann and Weikum (2010), Virtuoso), graph database management systems (GDBMSes) (Kankamamge et al. (2017), neo4j), XML databases (Gou and Chirkova (2007)), or extensions of relational databases that support graphs (Färber et al. 2012), also process other types of queries over graph-structured data. For example, some GDBMSes and RDF triple stores have query languages, such as Cypher and SPARQL, which can express typical relational operations on data, such as group-by-and-aggregates, filters, projections, ordering, as well as graph-specific queries such as *shortest path queries*. We focus on subgraph queries and RPQs as they are two of the most commonly appearing graph-specific queries in these systems. We note that some systems, such as Pregel (Malewicz et al. 2010) and some linear algebra software, have APIs to express other graph algorithms, such as PageRank, algorithms to find connected components, or BFS traversals, which we also do not cover in this chapter.

The high-level pipeline in existing systems to process subgraph queries and RPQs follows the classic pipeline in RDBMSes and consists of two steps: (1) a logical query plan generation and optimization and (2) a physical query plan generation, which outputs a physical plan that is evaluated on the actual graph data. We focus more on techniques to generate logical plans, which we think depict more commonality across systems than physical plans which are highly system-dependent.

Subgraph Queries

We consider the most basic form of subgraph queries in which the goal is to find instances of a directed subgraph Q , without any filters on its nodes or edges, in an input graph G . As a running example, consider the triangle query in Fig. 1a. Any subgraph query Q can be equivalently written as a relational multiway self-join query as follows. We label each vertex in Q with a distinct attribute a_i . The equivalent multiway join query contains an edge (a_i, a_j) relation



Graph Query Processing, Fig. 1 Triangle query and example edge-at-a-time and vertex-at-a-time plans. (a) Triangle Query. (b) Edge-at-a-time Plan. (c) Vertex-at-a-time Plan

for each edge (a_i, a_j) in Q . For example, in Datalog syntax, the triangle query is equivalent to:

$$tri(a_1, a_2, a_3) :=$$

$$edge(a_1, a_2), edge(a_1, a_3), edge(a_2, a_3)$$

Each edge table is a replica of each other and contains each edge (u, v) in G . Therefore, we think of subgraph queries as complex self-joins of replicas of an edge table.

Numerous approaches to evaluating subgraph queries exist in literature. We review two of these approaches, which we refer to as *edge-at-a-time* and *vertex-at-a-time* approaches. Then, we review two techniques which handle two scalability issues: (1) query decompositions, which are used when Q is large, i.e., contains many edges and vertices and (2) compressed output representations, which are used when the number of output instances is large. Large input graphs are a third scalability issue and are often handled by resorting to distributed systems. We omit a detailed discussion of this topic.

Edge-at-a-time Approaches

One of the most common approaches to evaluate subgraph queries is to match Q in G one edge at a time. This approach is equivalent to binary join plans. For example, one can evaluate the triangle query by first matching two of the edges and finding open triangles in G and then matching the third edge and closing these open triangles. This is equivalent to performing the following binary joins:

$$\begin{aligned} open-tri(a_1, a_2, a_3) &:= \\ edge(a_1, a_2), edge(a_2, a_3) & \\ tri(a_1, a_2, a_3) &:= \\ open-tri(a_1, a_2, a_3), edge(a_3, a_1) & \end{aligned}$$

The partial subgraphs matched by this plan are shown in Fig. 1b. Many RDF triple stores (Neumann and Weikum 2010), GDBMSes (neo4j), research prototype systems (Lai et al. 2016, 2017), as well as most RDBMSes (The only exception we are aware of is the LogicBlox system (Aref et al. 2015)), use this approach. There is a rich body of work on picking binary join plans. Systems first enumerate a set of join plans to pick from, such as left-deep or bushy plans, and then pick an actual plan using standard techniques. For example, TwinTwigJoin (Lai et al. 2017) considers only left-deep plans, while RDF-3x (Neumann and Weikum 2010) also considers bushy plans, and both systems use cost-based techniques that assign a cost to each possible plan, e.g., by estimating the sizes of intermediate tables that a plan generates and pick the lowest cost plan.

Vertex-at-a-time Approaches

Another approach is to match Q in G one vertex-at-a-time. Suppose Q contains m vertices a_1, \dots, a_m . This approach first picks an order of these m vertices, which we assume for simplicity to be a_1, \dots, a_m . We first compute $\Pi_{a_1} Q$, so match only vertex a_1 of Q in G . Then, these partial matches are extended by one vertex to compute $\Pi_{a_1, a_2} Q$, so and so forth, until finally a_1, \dots, a_{m-1} partial matches are extended to

compute $\Pi_{a_1, \dots, a_m} Q = Q$. Partial subgraphs to a new vertex a_i by effectively intersecting all of the edges that are incident on a_i in Q from already matched vertices a_1, \dots, a_{i-1} . Figure 1c shows the partial subgraphs that are matched when evaluating the triangle query vertex-at-a-time.

Readers might find the difference between vertex-at-a-time and edge-at-a-time approaches superficial. Indeed depending on how the intersections are computed, these approaches can even be equivalent. As an example, suppose a “vertex-at-a-time” algorithm takes (a_1, a_2) partial matches, e.g., $(1, 2)$, and for each outgoing edge of the vertex matching a_2 , in our case vertex 2, checks if the destination of that edge has an edge back to a_1 , in our case vertex 1. This effectively computes the intersection of 2’s out- and 1’s in-neighbors but is also equivalent to the edge-at-a-time join plan in Fig. 1b. Surprisingly, if the intersections are done in a particular way, there can be significant differences between these approaches. In particular, when extending a partial match (a_1, \dots, a_{i-1}) to a_i , the cost of intersecting all of the adjacent neighbor lists of a_i must take time proportional to the smallest adjacency list size, *for each partial match* (This can be achieved, for example, by first checking the sizes of the adjacency lists that need to be intersected and starting intersecting the elements in the smallest list in others.). Recent theoretical results have shown that using such fast intersections can make vertex-at-a-time approaches asymptotically faster than edge-at-a-time approaches (Ngo et al. 2012). For example, one can construct input graphs with N edges such that any edge-at-a-time plan will take $\Omega(N^2)$ time to compute the triangle query, yet vertex-at-a-time approaches will take $O(N^{3/2})$ time. These results were shown in the context of the new *worst-case optimal* join algorithms, such as NPPR (Ngo et al. 2012) and Leapfrog Triejoin (Veldhuizen 2012) algorithms, which evaluate queries one attribute at a time by such fast intersections.

Vertex-at-a-time approaches are used by many systems and algorithms, such as the Turbo_{ISO} (Han et al. 2013) and VF₂ (Cordella et al. 2004) algorithms, which do not necessarily

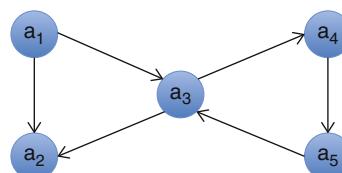
perform fast intersections, and the Empty-Headed (Aberger et al. 2016), LogicBlox (Aref et al. 2015) and Graphflow (Kankamamge et al. 2017) systems, which use worst-case optimal multiway join algorithms.

Query Decompositions

Queries that contain many vertices and edges are a common scalability challenge for evaluating subgraph queries. A popular technique used by both edge-at-a-time and vertex-at-a-time strategies is to decompose a query Q into smaller subqueries, SQ_1, \dots, SQ_k , evaluate each subquery separately, and then join them, instead of matching the edges or vertices of the query one by one. Bushy binary join plans are one type of decomposition. Decomposition has two main benefits:

- *Caching*: Decomposition avoids recomputing partial matched subgraphs SQ_i multiple times by caching and reusing them.
- *Filtering*: If the results of an SQ_i in the decomposed query is (partially) empty, we can (partially) avoid evaluating some SQ_j . As a simple example, consider the *bow tie query* in Fig. 2, which consists of an asymmetric triangle (a_1, a_2, a_3) and a symmetric triangle (a_3, a_4, a_5) sharing vertex a_3 . Suppose a vertex u is not part of any asymmetric triangles for a_3 , then we can skip computing the symmetric triangles of u for a_3 .

Decomposition is used by many systems and algorithms (Aberger et al. 2016; Lai et al. 2016; Neumann and Weikum 2010; Zhao and Han 2010). There are many possible decompositions of large queries, and different systems use different decompositions. For



Graph Query Processing, Fig. 2 Bow Tie Query

example, SEED (Lai et al. 2016) and SPath (Zhao and Han 2010), precompute and index several subgraphs, such as triangles or some paths, and pick decompositions that use these subgraphs. As another example, EmptyHeaded picks a special type of decomposition called *generalized hypertree decomposition* (GHD) and aims to have small number of edges in each SQ_i , which are evaluated using a vertex-at-a-time approach, but then joined using binary joins.

Compressed Output Representations

Another scalability challenge for systems and algorithms is the possibly very large outputs. On a graph with N edges, the number of matches of a query Q can be as large as $N^{\rho^*(Q)}$, where $\rho^*(Q)$ a quantity called the *fractional edge cover* of Q (Atserias et al. 2013). For example, there can be $\Omega(N^{3/2})$ triangles and $\Omega(N^3)$ bow ties. As a simple demonstrative example, an input graph that contains a clique of size 1000, will contain at least $\binom{1000}{5}$, roughly a quadrillion, many 5 cliques, which no system can effectively output in a reasonable time.

Several systems and algorithms handle this problem by using a compressed representation of the outputs for some queries. The techniques are varied, and we give high-level descriptions of several examples from literature. These compression techniques are closely related to the technique of *factorized databases* from the relational systems that have been used in several systems (Olteanu and Schleich 2016).

- *Clique compression:* SEED (Lai et al. 2016) detects maximal cliques in input graphs as a preprocessing step to avoid matching exponentially many instances of subgraphs in these cliques. Instead, these cliques are returned as output, and applications can enumerate the instances in them.
- *Cartesian Product-based Representation:* Reference (Qiao et al. 2017) proposes a technique to decompose Q into multiple subqueries (based on the vertex cover of Q) such that matches to the entire query can be represented as the union of a set of

Cartesian products of the matches of these subqueries. For example, a directed two-path query $edge(a_1, a_2), edge(a_2, a_3)$ can be represented as the union of Cartesian products of incoming and outgoing edges of each vertex.

G

Some systems and algorithms also compress the intermediate partial matches instead of the outputs. For example, TurboISO (Han et al. 2013) compresses two nodes a_i and a_j in Q that have the same neighborhoods into a single supernode (called *neighbor equivalence class*) and constructs a smaller query Q' . That is because if in an output a_i matches u and a_j matches v , there must be another output in which a_i matches v and a_j matches u . The algorithm evaluates Q' in a way that supernodes match sets of vertices in the graph. These sets are then expanded in different permutations to get outputs of Q . This technique effectively compresses the intermediate results that the algorithm generates.

Regular Path Queries

Regular path queries (RPQs) extend subgraph queries with nontrivial navigation on a graph. This navigation is performed by returning pairs of vertices in a graph such that paths between the vertices in a pair match a given regular expression. Formally, given a graph G as a tuple $\langle V, \Sigma, E \rangle$ for which V is a set of vertices, Σ is a set (alphabet) of labels, and $E \subseteq V \times \Sigma \times V$ is a set of directed, labeled edges; a regular path query Q is a tuple $\langle x, r, y \rangle$, where x and y are free variables which range over vertices and r is a regular expression. An *answer* to Q over a graph G is a vertex pair $\langle s, t \rangle \in V \times V$ such that there exists a path p from vertex s to vertex t such that it matches r . A *result* of a query Q on graph G is a collection of all answers of Q over G .

The semantics of regular path queries play an important role in their evaluation. Specifically, matching paths can be *simple* (no vertex is visited twice) or *arbitrary* (no such restriction). Further,

the result of the query can be *counting* (\forall), i.e., a query returns a *bag* of vertex pairs with each pair appearing in the result for each different matching path in a graph. On the other hand, the result can be *existential* (\exists , non-counting), i.e., a query returns a *set* of vertex pairs with each pair being an evidence that a matching path exists in a graph. Therefore, depending on the variations above, four types of semantical configurations for RPQs are distinguished: *simple*/ \exists , *simple*/ \forall , *arbitrary*/ \exists , and *arbitrary*/ \forall .

Intuitively, *simple* semantics have an implicit mechanism of dealing with cycles in a graph, while *arbitrary* configuration needs to deal with cycles explicitly to avoid unbounded computation. Further, \forall -semantics is very useful in SQL-style aggregation queries which \exists -semantics cannot handle.

It has been shown that *arbitrary*/ \exists is the only tractable configuration for a full fragment of regular expressions. Hence, a vast majority of systems which evaluate RPQs do so under *arbitrary*/ \exists semantics. SPARQL, a widely used graph query language, adopted *arbitrary*/ \exists RPQs in its *property paths* feature introduced as a part of SPARQL 1.1 specification. However, other semantical configurations are also used. For example, Cypher, a graph query language used in a popular neo4j graph database, is based on CHAREs (regular expressions with limited use of Kleene expressions) and semantics similar to *simple*/ \forall , but with paths with no repeated edges.

Graph Query Processing,

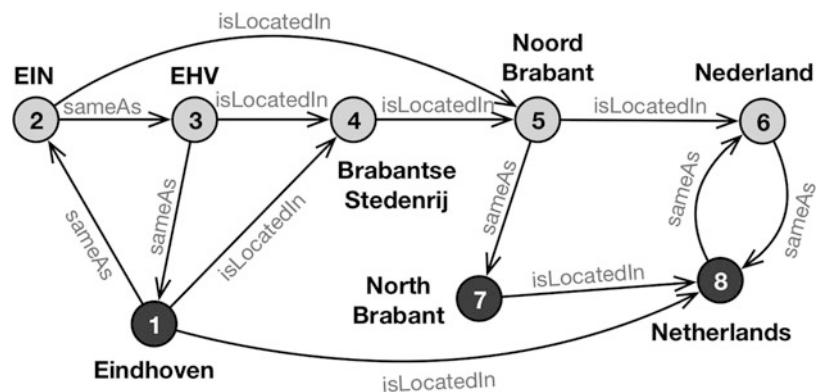
Fig. 3 Example of an interlinked English and Dutch DBpedia graph G_{DBP}

In the following subsections, we give a brief overview of different methods used in the evaluation of regular path queries. In literature, we identify three main groups of approaches: relational algebra-based, Datalog-based, and automata-based RPQ evaluation methods. Note that, due to lack of space, we do not discuss a wide body of work which uses specialized data structures (e.g., *reachability* indexes) in the evaluation of RPQs (e.g., see Gubichev et al. 2013; Fletcher et al. 2016). These methods are, in general, complementary to the methods we discuss below.

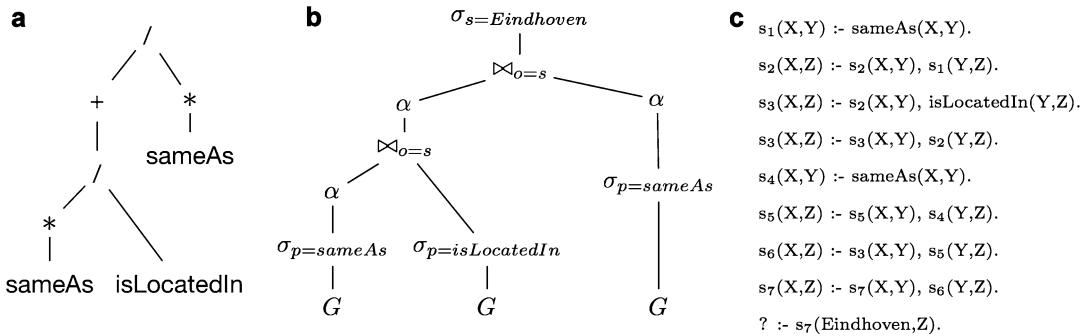
As a running example, consider a graph G_{DBP} and a query Q_{DBP} as shown in Fig. 3. Graph G_{DBP} contains information from both English and Dutch DBpedia datasets. Query Q_{DBP} aims to find all places p in both Dutch and English datasets in which the city of Eindhoven is located in. This query demonstrates the usefulness of RPQs in the navigation of unknown graphs, as it uses succinct but nontrivial Kleene transitive expressions to concurrently resolve both equivalence and geographical closures without prior knowledge of an underlying graph.

Relational Algebra and Datalog-Based Approaches

Authors of (Losemann and Martens 2012) proposed an RPQ evaluation algorithm based on a dynamic programming paradigm. Their approach



$$Q_{DBP} = \langle \text{Eindhoven}, (\text{isLocatedIn}/\text{sameAs}^*)^+/\text{sameAs}^*, p \rangle$$



Graph Query Processing, Fig. 4 A parse tree for a regular expression in Q_{DBP} , an α -RA tree and a Datalog program for Q_{DBP} . (a) A parse tree. (b) An α -RA tree (c) A Datalog program

G

can be modeled by the *relational algebra* extended by an additional operator α which computes the *transitive closure* of a given relation. This algebra, called α -RA (Agrawal 1988), can be then used to construct an α -RA expression tree (e.g., see Fig. 4b) based on the bottom-up traversal of a parse tree of the regular expression (Fig. 4a) in a given query.

Many GDBMSes implement RPQ evaluation by adapting the α -RA approach since it models relational techniques well and, thus, can be easily and efficiently implemented in most relational databases which support SQL-style recursion (e.g., PostgreSQL, Oracle, IBM DB2, and others) and relational triple stores (e.g., Virtuoso, IBM DB2 RDF store, and others). Evaluation based on α -extended relational algebra is also considered in (Dey et al. 2013; Yakovets et al. 2013).

Native support of recursion makes Datalog a suitable language to express regular path queries. For example, authors of (Consens et al. 1995) propose a simple strategy to convert an α -RA tree into a Datalog program (e.g., as shown in Fig. 4c). Then, this program can be submitted to any Datalog-compliant engine (e.g., LogicBlox) in order to evaluate a given RPQ.

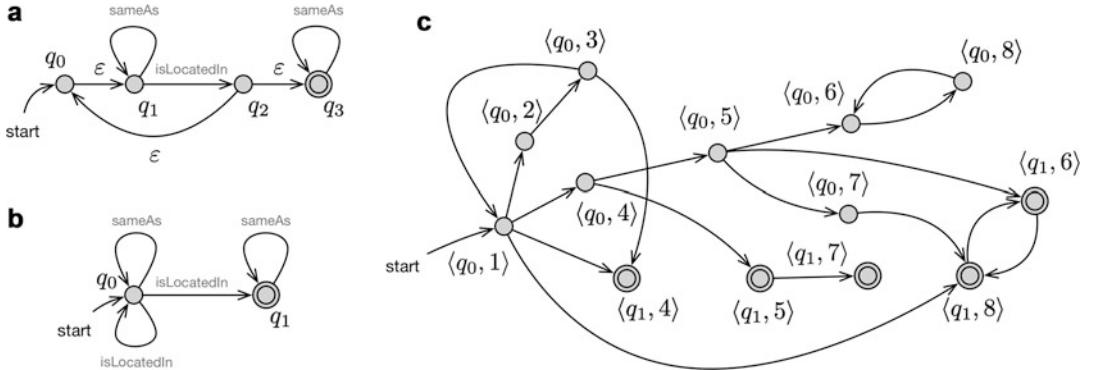
Note, however, that both α -RA tree and a corresponding Datalog program obtained by the translation procedure are not necessarily the most efficient as all the issues related to query planning apply to the tree and its translation as well (e.g., translation methods, join orders, query rewrites,

sideways information passing, and many others) (Consens et al. 1995; Yakovets et al. 2016).

Finite Automata-Based Approaches

Research on regular expressions, of course, well precedes the introduction of regular path queries. Specifically, regular expressions have been introduced in formal languages theory as a notation for patterns which *generate* words over a given alphabet. The dual problem to generation is *recognition* – and *finite state automata* (FA) are the recognizers for regular expressions. Specifically, given a regular expression r , there exists a number of methods to construct a corresponding automaton which recognizes the language of r . Note that there exist infinitely many FA which can recognize r , and many approaches exist which aim to construct an automaton deemed efficient for a particular scenario, e.g., finite automata obtained by using *minimization*.

Implementation of a **G+** query language (Mendelzon and Wood 1995) includes the first method which uses FA for evaluation of regular path queries on graphs. The algorithm works as follows. First, an automaton AQ_{DBP} (e.g., see Fig. 5a) which recognizes given regular expression r is constructed by using standard methods (e.g., Thompson's construction (Thompson 1968) and then, optionally, minimized (Fig. 5b). Then, a graph database G_{DBP} is converted into finite automaton AG_{DBP} with graph vertices becoming automaton states and



Graph Query Processing, Fig. 5 Example of automata for Q_{DBP} and a corresponding product automaton. (a) An ϵ -NFA for Q_{DBP} . (b) A minimized NFA for Q_{DBP} . (c) A product automaton $P_{DBP} = A_{Q_{DBP}} * A_{G_{DBP}}$

graph edges becoming automaton transitions in $A_{G_{DBP}}$. Given $A_{Q_{DBP}}$ and $A_{G_{DBP}}$, a product automaton $P_{DBP} = A_{Q_{DBP}} \times A_{G_{DBP}}$ (Fig. 5c) is constructed. Finally, automaton P_{DBP} is then tested for non-emptiness by checking whether any accepting state(s) can be reached from the starting state(s). Then, pairs of starting and corresponding reachable accepting states in P_{DBP} form the answer pairs for Q_{DBP} . The idea of using FA in evaluation of RPQs is used in Kochut and Janik (2007), Koschmieder and Leser (2012), Mendelzon and Wood (1995), Pérez et al. (2010), Yakovets et al. (2016), and Zauner et al. (2010). We briefly summarize some of these approaches below.

To avoid construction of a full product automaton, authors of Kochut and Janik (2007) propose to evaluate RPQs by running a bidirectional breadth-first search (biBFS) in the graph. Specifically, given a *reachability* query (i.e., an RPQ where variables are bound to constants on both ends), two FA are constructed. First automaton recognizes the regular language defined by the original expression, and the second automaton accepts the reversed language, which is also regular. The algorithm uses the steps of the biBFS to expand the search frontiers and to connect paths. Before each vertex in a graph is placed on the search frontier for the next expansion, the check is performed whether the partial path leading up to this vertex is not rejected by the appropriate (direct or reverse) automaton.

Authors of Koschmieder and Leser (2012) present an RPQ evaluation method which executes the search simultaneously in the graph and the automaton. This is achieved by exploring the graph using a BFS while marking the vertices in the graph with the corresponding states in the automaton. Essentially, in this method, edges in the graph are traversed only if the corresponding transition in the automaton allows it. This way, the construction of a full product automaton is avoided, and the automaton essentially acts as an evaluation plan for a given RPQ.

WaveGuide system (Yakovets et al. 2016) takes the idea of using an FA as an evaluation plan further. Authors show that effective plan spaces that result from FA-based and α -RA-based approaches are, in fact, incomparable. Hence, to find the best plan, both implicit plan spaces have to be considered. With this in mind, authors design a cost-based optimizer for RPQs which subsume and extend FA and α -RA-based approaches. Query plans in WaveGuide are based on the notion of *wavefronts* – stratified automata extended with seeds, append, prepend, and view transitions.

Research Directions

One important research direction in evaluating subgraph queries is understanding when the vertex-at-a-time algorithms perform better

than edge-at-a-time algorithm. For example, some early work in the area Nguyen et al. (2015) suggest that the worst-case optimal join algorithms outperform some systems that use binary join plans for some cyclic queries, but the topic requires a more thorough study.

Often applications do not just find subgraphs but perform other computations on the found subgraph outputs. Studying what kind of queries can be further executed on these outputs, when the outputs are compressed, is another research direction.

Finally, while it has been shown that RPQs allow for a rich plan space, it is still a challenge to generate *good* query evaluation plans with lowest execution costs. More research is needed to come up with accurate cost and cardinality estimation techniques used in graph query processing. This task is further complicated by the limited and often correlated structure of graph data.

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Path Navigation](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)
- ▶ [Graph Representations and Storage](#)
- ▶ [Parallel Graph Processing](#)

References

- Aberger CR, Tu S, Olukotun K, Ré C (2016) Empty-Headed: a relational engine for graph processing. In: SIGMOD
- Agrawal R (1988) Alpha: an extension of relational algebra to express a class of recursive queries. IEEE TSE 14(7):879–885
- Aref M, ten Cate B, Green TJ, Kimelfeld B, Olteanu D, Pasalic E, Veldhuizen TL, Washburn G (2015) Design and implementation of the logicblox system. In: SIGMOD
- Atserias A, Grohe M, Marx D (2013) Size bounds and query plans for relational joins. SIAM J Comput 42(4):1737–1767
- Consens MP, Mendelzon AO, Vista D, Wood PT (1995) Constant propagation versus join reordering in datalog. In: Rules in database systems. Springer, Berlin, Heidelberg, pp 245–259
- Cordella L, Foggia P, Sansone C, Vento M (2004) A (sub)graph isomorphism algorithm for matching large graphs. TPAMI 26(10):1367–1372
- Dey S, Cuevas-Vicentín V, Köhler S, Gribkoff E, Wang M, Ludäscher B (2013) On implementing provenance-aware regular path queries with relational query engines. In: Proceedings of the joint EDBT/ICDT 2013 workshops. ACM, pp 214–223
- Färber F, Cha SK, Primsch J, Bornhävd C, Sigg S, Lehner W (2012) SAP HANA database: data management for modern business applications. SIGMOD Rec 40(4): 45–51
- Fletcher GH, Peters J, Poulovassilis A (2016) Efficient regular path query evaluation using path indexes. In: Proceedings of the 19th international conference on extending database technology
- Gou G, Chirkova R (2007) Efficiently querying large XML data repositories: a survey. TKDE 19(10): 1381–1403
- Gubichev A, Bedathur SJ, Seufert S (2013) Sparqling Kleene: fast property paths in RDF-3X. In: Workshop on graph data management experiences and systems. ACM, pp 14–20
- Han WS, Lee J, Lee JH (2013) Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In: SIGMOD
- Kankanamge C, Sahu S, Mhedbhi A, Chen J, Salihoglu S (2017) Graphflow: an active graph database. In: SIGMOD
- Kochut KJ, Janik M (2007) SPARQLeR: extended SPARQL for semantic association discovery. In: The semantic web: research and applications. Springer, Berlin, pp 145–159
- Koschmieder A, Leser U (2012) Regular path queries on large graphs. In: Scientific and statistical database management. Springer, Berlin/Heidelberg, pp 177–194
- Lai L, Qin L, Lin X, Zhang Y, Chang L (2016) Scalable distributed subgraph enumeration. In: VLDB
- Lai L, Qin L, Lin X, Chang L (2017) Scalable subgraph enumeration in MapReduce: a cost-oriented approach. VLDB J 26(3):421–446
- Losemann K, Martens W (2012) The complexity of evaluating path expressions in SPARQL. In: Proceedings of the 31st symposium on principles of database systems. ACM, pp 101–112
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: SIGMOD
- Mendelzon A, Wood P (1995) Finding regular simple paths in graph databases. SIAM J Comput 24(6): 1235–1258
- Neumann T, Weikum G (2010) The RDF-3X engine for scalable management of RDF data. VLDB 19:91–113
- Ngo HQ, Porat E, Ré C, Rudra A (2012) Worst-case optimal join algorithms. In: PODS
- Nguyen D, Aref M, Bravenboer M, Kollias G, Ngo HQ, Ré C, Rudra A (2015) Join processing for graph patterns: an old dog with new tricks. In: GRADES workshop

- Olteanu D, Schleich M (2016) Factorized databases. SIGMOD Rec 45(2):5–16
- Pérez J, Arenas M, Gutierrez C (2010) nSPARQL: a navigational language for RDF. Web Semant Sci Serv Agents World Wide Web 8(4):255–270
- Qiao M, Zhang H, Cheng H (2017) Subgraph matching: on compression and computation. VLDB 11(2):17–188
- Thompson K (1968) Regular expression search algorithm. Commun ACM 11(6):419–422
- Veldhuizen TL (2012) Leapfrog Triejoin: a worst-case optimal join algorithm. CoRR abs/1210.0481
- Yakovets N, Godfrey P, Gryz J (2013) Evaluation of SPARQL property paths via recursive SQL. In: AMW
- Yakovets N, Godfrey P, Gryz J (2016) Query planning for evaluating SPARQL property paths. In: SIGMOD, San Francisco, pp 1875–1889
- Zauner H, Linse B, Furche T, Bry F (2010) A RPL through RDF: expressive navigation in RDF graphs. In: Web reasoning and rule systems. Springer, Heidelberg, pp 251–257
- Zhao P, Han J (2010) On graph query optimization in large networks. VLDB 3(1–2):340–351

Graph Representations and Storage

Marcus Paradies^{1,3} and Hannes Voigt²

¹SAP SE, Walldorf, Germany

²Dresden Database Systems Group, Technische Universität Dresden, Dresden, Germany

³SAP SE, Berlin, Germany

Definitions

Graph representation concerns the layout of graph data in the linearly addressed storage of computers (main memory, SSD, hard disk, etc.). General objectives for graph representations are (1) to use little storage space (space-efficiency) while (2) allowing operations and queries on the graph to be executed in a short time (time-efficiency). Hence, every graph representation technique is subject to a specific space–time trade-off. It depends on the particular scenario (data, workload, available resources, etc.) whether the space–time trade-off of a certain representation technique is worthwhile.

Overview

Graph data comes with a large variation of graph data models, with RDF and the property graph model (PGM) being the most prominent ones. The specific graph representation depends on the data model. We introduce the most important concepts for graph representations as data-model-agnostic as possible. The discussion is structured in (1) primary representation of the graph adjacency, (2) primary representation of properties (i.e., data) associated with graph objects (nodes and edges), and (3) indexing, i.e., the secondary representation of graph data.

Key Research Findings

Adjacency Representation

The adjacency representation concerns only the graph topology, i.e., the structure of the graph, specifically the pairs of nodes that are connected by an edge. Here, we assume a simple, directed graph $G(V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges.

Adjacency Matrix. An adjacency matrix, the most basic storage structure for graphs, stores the topology as a binary matrix representing each edge in the graph by an individual bit stored at coordinates (u, v) for vertices $u, v \in V$ (Cormen et al. 2001). With a quadratic space complexity of $\mathcal{O}(|V|^2)$, an adjacency matrix is only space-efficient for very dense graphs. However, often graphs are sparse in practice; therefore a pure adjacency matrix representation is rather uncommon in graph data management and graph processing systems.

Adjacency List. Being more suitable for sparse graphs, an adjacency list stores for each vertex a linked list of adjacent vertices and has a space complexity of $\mathcal{O}(|V| + |E|)$. Technically, an adjacency list can be implemented either by a pointer array to represent vertices and a set of linked lists to represent adjacent vertices or by an array of arrays. While a list-based adjacency data structure allows higher in-place update rates, an array-based adjacency list exhibits a better CPU

cache utilization for retrieving the adjacency of a vertex through a strictly sequential and contiguous memory access pattern. This is particularly important when the graph is stored in memory.

CSR. The *compressed sparse row* (CSR) data structure is a variation of the adjacency list. It has been developed in the context of sparse matrix multiplications and is widely used in graph processing systems (Gustavson 1978). The CSR data structure represents the graph topology as an *edge array* consisting of the target vertices of all edges in sequential order. To retrieve the adjacency of a single vertex, the *offset array* is used to compute the boundary values of the adjacency. In comparison to the adjacency list and the adjacency matrix, a CSR is more compact since the entire topology is represented in a single, large, continuous chunk of memory. The downside is that adding and removing vertices/edges is expensive and results in a partial reconstruction of the CSR to maintain the ordering. A detailed experimental analysis of several graph representations is given in Blandford et al. (2004).

Adjacency Compression. Various works consider succinct representations of the graph topology (Boldi and Vigna 2004; Apostolico and Drovandi 2009; Claude and Navarro 2010a,b; Boldi et al. 2011; Grabowski and Bieniecki 2011). Brisaboa et al. (2014) that provides a good overview. Currently, the most compact graph representation that still allows fast edge existence checks is based on k^2 -ary tree structures (Brisaboa et al. 2014). Specifically, this representation exploits the sparseness of the adjacency matrix and stores a graph in a compact tree structure of height $\lceil \log_k n \rceil$. The tree consists of nodes, where each node has a single bit assigned indicating whether there is at least one edge in the corresponding submatrix. A node assigned zero means that it has no child nodes and all elements in the submatrix are zero. The authors use the parameter k to configure the height of the tree at the cost of larger internal nodes with more child nodes. Internally, a k^2 -tree is represented by two bit sets, one for the internal nodes (except leaves) and one for the leaves

only. Bits are assigned according to a level-wise traversal of the tree, i.e., first all bits of level one, then all bits of level two, and so on. Although the data structure provides a concise representation of the graph topology, multi-relational graphs, i.e., multiple edges between a pair of vertices, are not supported. Additionally, the k^2 -ary tree is not designed for dynamic graphs with frequent edge insertions/deletions as each of these operations might trigger a rewriting of multiple internal nodes in the tree.

G

Adaptive Bit Set Representations. Aberger et al. (2016) propose to represent edges in an immutable and adaptive CSR data structure, which allows storing a single neighborhood either in a dense or a sparse representation. The actual representation is selected by an optimizer based on collected statistics about the density and the cardinality of the neighborhood set. The authors define the *density* of a neighborhood set as the cardinality of the set divided by the value range of the set. Real-world graphs tend to have a large skew in their neighborhood density distribution.

Property Representation

Property representation concerns approaches to represent all triples of the form (s, p, o) , where s is an entity, p is a property key, and o is a value. These properties can be node attributes and edge attributes. However, outgoing edges (or their label) can be treated as properties of a node. Likewise, source and target node of an edge can be treated as properties of an edge. Hence, approaches for property representation are often also used for adjacency representation. In general, techniques for the property representation are becoming more relevant in the presence of attributed or labeled graphs, as prevailing in RDF and the PGM.

Triple Table. A generic, straightforward approach for storing triple data is a relational data representation. A triple set can be easily stored in a single, universal *triple table* with three columns, S, P, and O (Chong et al. 2005). This approach is very common for RDF data. Typically, a triple table is combined with dictionary compression

to reduce the storage overhead implied by reoccurring, long identifiers. The addition of indexes allows for faster lookups. The downside is that a triple table requires many join operations to read an entity and its accompanying properties.

Universal Table. Alternatively, triples can also be stored in a universal table. The universal table maps each distinct property key to a separate column, and each object is represented as a single record in the table. Precisely, each row represents an s , each column a p , and each cell o so that $s[p] = o$ if there is triple (s, p, o) and $s[p] = \text{NULL}$ otherwise. This approach is very common for PGM data. In typical PGM data, nodes and edges do not share many property keys so that it is beneficial to use two universal tables, a *vertex table* and an *edge table* (Rudolf et al. 2013; Paradies et al. 2015). This approach allows the join-free reconstruction of a vertex/an edge through a selection and a subsequent attribute projection. Both, vertices and edges, can have an arbitrary number of attributes, though. Hence, universal tables require an efficient handling of NULL values.

Emerging Schemas. A solution that reduces the number of joins without introducing too many NULL values tries to exploit the schema inherent in the graph data. Since graph data usually has no upfront defined and rigid database schema, finding a suitable normalized database design for it is a nontrivial and tedious task. The simple *property-class* pattern partitions entities by class (RDF) or label (PGM) (Abadi et al. 2007). More general *property tables* cluster properties that frequently occur together into a table. Individual entities may be represented in many of these property tables, depending on which property cluster they fully or partially instantiate (Abadi et al. 2007). For a given dataset, a good set of property tables can be found automatically. Approaches are (1) utilizing standard clustering algorithms such as k-means (MacQueen 1967), cf. (Chu et al. 2007), (2) starting from every distinct property set and merge infrequent ones (Pham et al. 2015), or (3) partitioning entities on the fly based on

their schema similarity into small fix-sized partitions, such as memory blocks or disk pages (Herrmann et al. 2014a,b). The latter approach is lightweight, does not require any upfront computation, and can readapt if the inherent schema changes over time.

Schema Hashing. Another technique for NULL value reduction in universal tables is *schema hashing* (Bornea et al. 2013). Here, the universal table consists of n column groups. One column group consists of a **property** column and a **value** column, i.e., one column group represents a predicate–object pair or a property key–value pair from RDF or PGM, respectively. A hash function maps a predicate/property to one of the n column groups. With a well-designed hash function, properties are packed densely into the table with only few collisions. In case of a collision, the entity has to spill into an additional row. The same principle can be applied to labeled edges; then a column group consists of three columns, **edge-id**, **label**, and **target-vertex** (Sun et al. 2015). In this way, the complete neighborhood of a vertex can be represented in a single row.

Separate Property Storage. Instead of storing schema-flexible properties in the same data structure as the graph topology, some systems store properties in a separate data structure. For instance, it is possible to utilize a document store or the JSON (Crockford 2006) extension of a relational DBMS, to store all properties of an entity as a single JSON document in a single table cell of a separate vertex or edge attribute table (Sun et al. 2015). Another approach is to separate properties into a key–value table. By doing so, vertices and edges can be stored as fixed-length records in a vertex and an edge table, respectively. Fixed-length records can be directly addressed, which allows $O(1)$ access complexity to adjacent vertices.

Indexing

Exhaustive Indexing. State of the art in RDF representation is sixfold indexing, which

exhaustively indexes every permutation of the three triple elements, S, P, and O (Neumann and Weikum 2008; Weiss et al. 2008). In total, this approach results in six indexes: SPO, SOP, PSO, POS, OSP, and OPS. Likewise, every graph can be indexed on the source (S) and target (T) of the edges exhaustively by storing the adjacency once in forward order (ST) and once in reverse order (TS) (Hauck et al. 2015; Shun and Blelloch 2013). Exhaustive indexing of graphs is feasible since dictionary and delta compression allow for compact index representations. Furthermore, exhaustive indexing provides considerable performance gains since it facilitates direct neighborhood access over incoming as well as outgoing edges and efficient sort merge join operations on every combination of incident edges.

Aggregate Indexing. Additionally, aggregate indexes provide direct access to degree information (Neumann and Weikum 2008; Shun and Blelloch 2013). An aggregate index on S provides out-degrees; one on T provides in-degrees. RDF systems can maintain up to nine aggregate indexes: SP, SO, PS, PO, OS, OP, S, P, and O. Aggregate indexes can be stored even more compactly than regular graph indexes. They are useful for queries that project to a subsets of query variables, where only the existence of not fully qualified triples needs to be checked. Further aggregate indexes provide useful statistics, which allow choosing more efficient query plans (Neumann and Weikum 2008) or processing strategies (Shun and Blelloch 2013).

Reachability and Distance Indexing. A large body of work concerns index structures supporting reachability and distance queries. Given a graph, reachability queries answer whether for a given pair of nodes u and v there is a path between them and distance queries return the minimum length of the path connecting a given pair of nodes u and v . Distance is measured either in number of edges or as sum of edge weights along the path. Most approaches index a graph by

computing a label for each node, which encodes the reachability or the distance to other nodes in the graph.

For instance, a classic approach computes *2-hop labels*, where each node v is assigned a label $L(v) = (L_{\text{in}}(v), L_{\text{out}}(v))$ such that $L_{\text{in}}(v)$ and $L_{\text{out}}(v)$ are a set of nodes from which v can be reached and a set of nodes that are reachable from v , respectively. The labels are chosen in such a way that for any given pair of nodes u and v , the intersection of $L_{\text{out}}(u)$ and $L_{\text{in}}(v)$ is not empty if and only if v is reachable from u . The approach can be generalized to distance queries by storing a distance $d(x, v)$ and $d(v, x)$ together with every x in $L_{\text{in}}(v)$ and $L_{\text{out}}(v)$, respectively, such that the minimum of $d(u, x) + d(x, v)$ among all $x \in L_{\text{out}}(u) \cap L_{\text{in}}(v)$ is equal to (or sufficiently approximates) the distance between u and v . The challenge is to efficiently compute a minimal set of such labels.

For reachability indexes, another common approach uses ranges to encode reachable nodes in a label. Therefore, all nodes are assigned a label identifier by (1) collapsing all strongly connected components into single nodes, (2) traversing the resulting graph depth-first from some (artificially introduced) root node, and (3) numbering all nodes in post-order. Then, all nodes x reachable from a given node v can be succinctly listed in $L(v)$ as a set of ranges of label identifiers so that a node u is reachable from v if $u \in L(v)$ (Yu and Cheng 2010). The size of these labels can be further reduced by just storing approximations that can answer most negative reachability query (resulting in *false*) and efficiently steer a depth-first search in the graph for the remaining queries.

For more details on reachability and distance indexing, we refer the reader to surveys on reachability queries (Yu and Cheng 2010; Su et al. 2017) and distance queries (Sommer 2014; Bast et al. 2016).

Structural Indexing. It is also possible to store a summarization of a graph, which can help to speed up conjunctive and regular path queries. For this topic we refer the reader to the entry on graph indexing and synopses.

Partitioning of Big Graphs

Very large graphs can be partitioned to distribute processing or a query workload across multiple workers (e.g., machines in a cluster or CPUs in a NUMA system). The objective is to maximize parallel execution of the given workload and therefore equally distribute the load to a maximum number of workers while minimizing the synchronization needs between these workers. Obviously, these are conflicting goals. Additionally, the optimal partitioning strategy depends on the workload, the graph topology, the hardware setup (worker capabilities, communication cost, etc.), and the processing model.

Possible strategies can be categorized by the *partitioning criterion*, the *balancing criterion* they apply (Krause et al. 2017), and the *redundancy level* they permit. The partitioning criterion is the domain whose elements are considered atomic in the partitioning. The balancing criterion is the domain of which the resulting partitions should contain a balanced number of elements. For both criteria, domains are either edges, vertices, or components (connected subgraphs). The redundancy level is the amount of overlap that is permitted between partitions. The possible redundancy level ranges from completely disjoint partitions to full redundancy, where each worker receives a complete copy of the graph. For each strategy there are multiple algorithms. For an overview, we refer the reader to the entry on graph partitioning.

Given the large number of possibilities, it is not obvious which is the optimal partitioning strategy for a given setup. In consequence, many systems rely on hash partitioning of nodes (or edges), which is the simplest of all options. Any more sophisticated solution should perform better than hash partitioning to justify extra effort. Various works improve hash partitioning of node on graphs with skewed degree distribution by hash partitioning low-degree nodes and hash partitioning the edges high-degree nodes, e.g., LeBeane et al. (2015) and Chen et al. (2015). Also adaptive strategies have been proposed, e.g., Khayyat

et al. (2013). As of now, however, no graph partitioning strategy showed to be superior for a wide range of use cases (Verma et al. 2017; Krause et al. 2017), neither are cause–effect relations between graph partitioning and performance of parallel system clearly understood.

Examples of Applications

Graph data management systems implement the discussed graph representation techniques in a variety of combinations. For many systems, the implemented graph representation is not publicly known; hence the following is just a sample and not meant to be complete.

RDF Database Systems. RDF-3X (Neumann and Weikum 2008) builds on exhaustively indexed triple table storage with a set of lightweight compression techniques, namely, dictionary compression, delta encoding, and variable byte encoding to keep each index small and space-efficient. Additionally, RDF-3X uses aggregate indexes. Virtuoso (Erling and Mikhalov 2007), Hexastore (Weiss et al. 2008), and RDFox (Nenov et al. 2015) build on an exhaustively indexed triple table with slight variations in how the exhaustively is precisely implemented.

PGM Database Systems. Neo4j stores the property graph essentially in three tables, a node and an edge table each with a fixed record length and a triple table for node and edge properties. SAP HANA Graph (Rudolf et al. 2013) represents a graph in a pair of vertex and edge table following the universal table approach. Additionally, SAP HANA Graph employs CSR structures to exhaustively index the adjacency. IBM’s SQLGraph utilizes schema hashing for edges and a separated property storage by leveraging the JSON extension of a relational DBMS for property storage (Sun et al. 2015). ArangoDB builds a PGM store on top of a JSON document store, storing nodes and edges as JSON documents.

Cross-References

- ▶ Graph Data Models
- ▶ Graph Partitioning: Formulations and Applications to Big Data
- ▶ Parallel Graph Processing

References

- Abadi DJ, Marcus A, Madden S, Hollenbach KJ (2007) Scalable semantic web data management using vertical partitioning. In: VLDB. ACM, pp 411–422
- Aberger CR, Tu S, Olukotun K, Ré C (2016) EmptyHeaded: a relational engine for graph processing. In: SIGMOD, pp 431–446. <https://doi.org/10.1145/2882903.2915213>
- Apostolico A, Drovandi G (2009) Graph compression by BFS. Algorithms 2(3):1031–1044. <https://doi.org/10.3390/a2031031>
- Bast H, Delling D, Goldberg AV, Müller-Hannemann M, Pajor T, Sanders P, Wagner D, Werneck RF (2016) Route planning in transportation networks. In: Algorithm engineering – selected results and surveys, vol 9220, pp 19–80. https://doi.org/10.1007/978-3-319-49487-6_2
- Blandford DK, Blelloch GE, Kash IA (2004) An experimental analysis of a compact graph representation. In: ALENEX. SIAM, pp 49–61
- Boldi P, Vigna S (2004) The webGraph framework I: compression techniques. In: WWW. ACM, pp 595–602. <https://doi.org/10.1145/988672.988752>
- Boldi P, Rosa M, Santini M, Vigna S (2011) Layered label propagation: a multiResolution coordinate-free ordering for compressing social networks. In: WWW. ACM, pp 587–596. <https://doi.org/10.1145/1963405.1963488>
- Bornea MA, Dolby J, Kementsietsidis A, Srinivas K, Dantressangle P, Udrea O, Bhattacharjee B (2013) Building an efficient RDF store over a relational database. In: SIGMOD. ACM, pp 121–132. <https://doi.org/10.1145/2463676.2463718>
- Brisaboa NR, Ladra S, Navarro G (2014) Compact representation of web graphs with extended functionality. Inf Syst 39:152–174. <https://doi.org/10.1016/j.is.2013.08.003>
- Chen R, Shi J, Chen Y, Chen H (2015) PowerLyra: differentiated graph computation and partitioning on skewed graphs. In: EuroSys. ACM, pp 1:1–1:15. <https://doi.org/10.1145/2741948.2741970>
- Chong EI, Das S, Eadon G, Srinivasan J (2005) An efficient SQL-based RDF querying scheme. In: VLDB. ACM, pp 1216–1227
- Chu E, Beckmann JL, Naughton JF (2007) The case for a wide-table approach to manage sparse relational data sets. In: SIGMOD. ACM, pp 821–832. <https://doi.org/10.1145/1247480.1247571>
- Claude F, Navarro G (2010a) Extended compact web graph representations. In: Algorithms and applications, essays dedicated to Esko Ukkonen on the occasion of his 60th birthday. Springer, pp 77–91. https://doi.org/10.1007/978-3-642-12476-1_5
- Claude F, Navarro G (2010b) Fast and compact web graph representations. ACM Trans Web 4(4):16:1–16:31. <https://doi.org/10.1145/1841909.1841913>
- Cormen TH, Leiserson CE, Rivest R, Stein C (2001) Introduction to algorithms, 2nd edn. MIT Press, Cambridge
- Crockford D (2006) The application/json media type for JavaScript object notation (JSON), RFC 4627. <http://tools.ietf.org/html/rfc4627>
- Erling O, Mikhailov I (2007) RDF support in the virtuoso DBMS. In: CSSW, GI, vol 113, pp 59–68
- Grabowski S, Bieniecki W (2011) Merging adjacency lists for efficient web graph compression. In: ICMMI. Springer, pp 385–392. https://doi.org/10.1007/978-3-642-23169-8_42
- Gustavson FG (1978) Two fast algorithms for sparse matrices: multiplication and permuted transposition. Trans Math Softw 4(3):250–269. <https://doi.org/10.1145/355791.355796>
- Hauck M, Paradies M, Fröning H, Lehner W, Rauhe H (2015) Highspeed graph processing exploiting main-memory column stores. In: Euro-Par, pp 503–514. https://doi.org/10.1007/978-3-319-27308-2_41
- Herrmann K, Voigt H, Lehner W (2014a) Cinderella – adaptive online partitioning of irregularly structured data. In: SMDB. IEEE Computer Society, pp 284–291. <https://doi.org/10.1109/ICDEW.2014.6818342>
- Herrmann K, Voigt H, Lehner W (2014b) Online horizontal partitioning of heterogeneous data. It – Inf Technol 56(1):4–12. <https://doi.org/10.1515/itit-2014-1015>
- Khayyat Z, Awara K, Alonazi A, Jamjoom H, Williams D, Kalnis P (2013) Mizan: a system for dynamic load balancing in large-scale graph processing. In: EuroSys. ACM, pp 169–182. <https://doi.org/10.1145/2465351.2465369>
- Krause A, Kissinger T, Habich D, Voigt H, Lehner W (2017) Partitioning strategy selection for in-memory graph pattern matching on multiprocessor systems. In: Euro-Par
- LeBeane M, Song S, Panda R, Ryoo JH, John LK (2015) Data partitioning strategies for graph workloads on heterogeneous clusters. In: SC. ACM, pp 56:1–56:12. <https://doi.org/10.1145/2807591.2807632>
- MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: BSMSP. University of California Press, vol 1, pp 281–297
- Nenov Y, Piro R, Motik B, Horrocks I, Wu Z, Banerjee J (2015) RDFox: a highly-scalable RDF store. In: ISWC. Springer, vol 9367, pp 3–20. https://doi.org/10.1007/978-3-319-25010-6_1
- Neumann T, Weikum G (2008) RDF-3X: a RISC-style engine for RDF. PVLDB 1(1):647–659
- Paradies M, Lehner W, Bornhövd C (2015) GRAPHITE: an extensible graph traversal framework for relational database management systems. In: SS-

- DBM. ACM, pp 29:1–29:12. <https://doi.org/10.1145/2791347.2791383>
- Pham M, Passing L, Erling O, Boncz PA (2015) Deriving an emergent relational schema from RDF data. In: WWW. ACM, pp 864–874. <https://doi.org/10.1145/2736277.2741121>
- Rudolf M, Paradies M, Bornhövd C, Lehner W (2013) The graph story of the SAP HANA database. In: BTW, GI, vol 214, pp 403–420
- Shun J, Blelloch GE (2013) Ligra: a lightweight graph processing framework for shared memory. In: PPoPP, pp 135–146. <https://doi.org/10.1145/2442516.2442530>
- Sommer C (2014) Shortest-path queries in static networks. ACM Comput Surv 46(4):45:1–45:31. <https://doi.org/10.1145/2530531>
- Su J, Zhu Q, Wei H, Yu JX (2017) Reachability querying: can it be even faster? IEEE TKDE 29(3):683–697. <https://doi.org/10.1109/TKDE.2016.2631160>
- Sun W, Fokoue A, Srinivas K, Kementsietsidis A, Hu G, Xie GT (2015) SQLGraph: an efficient relational-based property graph store. In: SIGMOD. ACM, pp 1887–1901. <https://doi.org/10.1145/2723372.2723732>
- Verma S, Leslie LM, Shin Y, Gupta I (2017) An experimental comparison of partitioning strategies in distributed graph processing. PVLDB 10(5):493–504
- Weiss C, Karras P, Bernstein A (2008) Hexastore: sextuple indexing for semantic web data management. PVLDB 1(1):1008–1019
- Yu JX, Cheng J (2010) Graph reachability queries: a survey. In: Managing and mining graph data, vol 40. Springer, pp 181–215. https://doi.org/10.1007/978-1-4419-6045-0_6

Graph Summarization

► Graph OLAP

Graph Visualization

Yifan Hu¹ and Martin Nöllenburg²

¹Yahoo Research, Oath Inc., New York, NY, USA

²Institute of Logic and Computation, TU Wien, Vienna, Austria

Synonyms

Graph drawing; Graph layout; Network visualization

Definitions

Graph visualization is an area of mathematics and computer science, at the intersection of geometric graph theory and information visualization. It is concerned with visual representation of graphs that reveals structures and anomalies that may be present in the data and helps the user to understand and reason about the graphs.

Overview

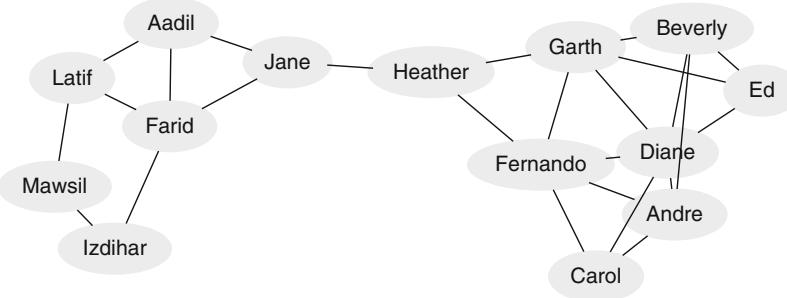
Graph visualization is concerned with visual representations of graph or network data. Effective graph visualization reveals structures that may be present in the graphs and helps the users to understand and analyze the underlying data.

A graph consists of nodes and edges. It is a mathematical structure describing relations among a set of entities, where a node represents an entity, and an edge exists between two nodes if the two corresponding entities are related.

A graph can be described by writing down the nodes and the edges. For example, this is a social network of people and how they relate to each other:

$\{Andre \leftrightarrow Beverly, Andre \leftrightarrow Diane, Andre \leftrightarrow Fernando, Beverly \leftrightarrow Garth, Beverly \leftrightarrow Ed, Carol \leftrightarrow Andre, Carol \leftrightarrow Diane, Carol \leftrightarrow Fernando, Diane \leftrightarrow Beverly, Diane \leftrightarrow Garth, Diane \leftrightarrow Ed, Farid \leftrightarrow Aadil, Farid \leftrightarrow Latif, Farid \leftrightarrow Izdihar, Fernando \leftrightarrow Diane, Fernando \leftrightarrow Garth, Fernando \leftrightarrow Heather, Garth \leftrightarrow Ed, Garth \leftrightarrow Heather, Heather \leftrightarrow Jane, Izdihar \leftrightarrow Mawsil, Jane \leftrightarrow Farid, Jane \leftrightarrow Aadil, Latif \leftrightarrow Aadil, Mawsil \leftrightarrow Latif\}.$

This social network tells us that “Farid” is a friend of “Aadil,” “Latif” is a friend of “Aadil,” and so on. However, this mathematical notation of the network does not convey immediately the structure of the network. On the other hand, Fig. 1 shows a visualization of this graph. We can see at a glance that this graph has two clusters. This example illustrates that graph visualization can give us an overall sense of the data. It reveals structures and anomalies and helps us to ask questions that can in turn be answered through interacting with the visualization itself or by



Graph Visualization, Fig. 1 Graph visualization of a small social network

G

writing algorithms to mine the data for evidence seen in the visualization.

In this chapter, a graph $G = (V, E)$ consists of a set of nodes (vertices) V and a set of edges E , which are pairs of nodes. Denote by $n = |V|$ and $m = |E|$ the number of nodes and edges, respectively. If there is an edge from node i to node j , we denote that as $i \rightarrow j$. If the graph is undirected, then we denote the edge as $i \leftrightarrow j$ and call i and j neighboring (or adjacent) nodes.

Node-Link Diagrams and Layout

Aesthetics

By far the most common type of graph layout is the so-called *node-link diagram* as seen in Fig. 1. Here nodes are represented by points or simple geometric shapes like ellipses or rectangles, whereas edges are drawn as simple curves linking the corresponding pair of nodes. In this chapter we restrict our attention to node-link diagrams; for alternative types of graph representations, see the survey of von Landesberger et al. (2011).

The algorithmic graph layout problem consists in finding node positions in the plane (or in three-dimensional space) and edge representations as straight lines or simple curves such that the resulting layout faithfully depicts the graph and certain aesthetic quality criteria are satisfied and optimized. We list the most common criteria. The influence of most of them on human graph reading tasks has been empirically confirmed (Purchase 1997).

- **crossings:** the fewer edge crossings the better (a layout without crossings exists only for *planar* graphs)

- **bends:** the fewer edge bends the better; ideally edges are straight-line
- **edge lengths:** use as uniform edge lengths as possible
- **angular resolution:** angles between edges at the same node should be large
- **crossing angles:** angles of pairs of crossing edges should be large
- **area and aspect ratio:** the layout should be as compact as possible
- **edge slopes:** few and regularly spaced edge slopes should be used (e.g., *orthogonal* layouts use only horizontal and vertical edges)
- **neighborhood:** neighbors of each node in the graph should be neighbors in the layout as well

The above list is not comprehensive, and there may be additional application-specific constraints and criteria or global aspects such as displaying symmetries. Moreover, some criteria may contradict each other. Typically only a subset of criteria is optimized by a graph layout algorithm, and trade-offs between different criteria need to be considered.

Key Research Findings

Undirected Graph Drawing

The layout algorithms and techniques presented in this section do not make use of edge directions, but they can obviously be applied to both undirected and directed graphs. Undirected graphs are often represented by node-link diagrams with

Algorithm 1 ForceDirectedAlgorithm(G, x, tol, K)

```

1 input: graph  $G = (V, E)$ , initial positions  $x$ , tolerance  $tol$ , and nominal edge length  $K$ 
2 set  $step$  = initial step length
3 repeat
4    $x^0 = x$ 
5   for ( $i \in V$ ) {
6      $f = 0$  //  $f$  is a 2/3D vector
7     for ( $j \leftrightarrow i, j \in V$ )  $f \leftarrow f + F_a(i, j)$  // attractive force, see equation (1)
8     for ( $j \neq i, j \in V$ )  $f \leftarrow f + F_r(i, j)$  // repulsive force, see equation (2)
9      $x_i \leftarrow x_i + step * (f / \|f\|)$  // update position of node  $i$ 
   }
10  until ( $\|x - x^0\| < tol * K$ )
11  return  $x$ 
```

straight-line edges. We denote by x_i the location of node i in the layout. Here x_i is a point in two- or three-dimensional Euclidean space.

Spring embedders (Eades 1984; Fruchterman and Reingold 1991; Kamada and Kawai 1989) are the most widely used layout algorithms for undirected graphs. They attempt to find aesthetic node placement by representing the problem as one of minimizing the energy of a physical system. The guiding principles are that nodes that are connected by an edge should be near each other, while no nodes should be too close to each other (cf. neighborhood aesthetic). Depending on the exact physical model, we could further divide the spring embedders into two types. For convenience, we call the first type spring-electrical model and the second type spring/stress model.

Spring-Electrical Model

This model was first introduced by Peter Eades (1984). A widely used variant, which is given below, is due to Fruchterman and Reingold (1991). The model is best understood as a system of springs and electrical charges; therefore we name this as the spring-electrical model, to differentiate the spring/stress model that relies on springs only, even though historically both are called spring embedders.

In this model, each edge is replaced by a spring with an ideal length of 0, which pulls nodes that share an edge together. At the same time, imagine that nodes have the same type of

electrical charges (e.g., positive) that push them apart. Specifically, there is an attractive spring force exerted on node i from its neighbor j , which is proportional to the squared distance between these two nodes:

$$F_a(i, j) = -\frac{\|x_i - x_j\|^2}{K} \frac{x_i - x_j}{\|x_i - x_j\|}, \quad i \leftrightarrow j, \quad (1)$$

where K is a parameter related to the nominal edge length of the final layout. The repulsive electrical force exerted on node i from *any* node j is inversely proportional to the distance between these two nodes:

$$F_r(i, j) = \frac{K^2}{\|x_i - x_j\|} \frac{x_i - x_j}{\|x_i - x_j\|}, \quad i \neq j. \quad (2)$$

The spring-electrical model can be solved with a force-directed procedure by starting from an initial (e.g., random) layout, calculating the combined attractive and repulsive forces on each node, and moving the nodes along the direction of the force for a certain step length. This process is repeated, with the step length decreasing every iteration, until the layout stabilizes. This procedure is formally stated in Algorithm 1.

The spring-electrical model as described by Eqs. 1 and 2 cannot be used directly for large graphs. The repulsive force exists on all pairs of nodes, so the computational complexity is quadratic in the number of nodes. Force

approximation techniques based on space decomposition data structure, such as the Barnes-Hut algorithm, can be used to approximate the repulsive forces efficiently (Tunkelang 1999; Quigley 2001; Hachul and Jünger 2004).

For large graphs, the force-directed algorithm, which uses the steepest descent process and repositions one node at a time to minimize the energy locally, is likely to be stuck at local minima, because the physical system of springs and electrical charges can have many local minimum configurations. This can be overcome by the multilevel technique. In this technique, a sequence of smaller and smaller graphs are generated from the original graph; each captures the essential connectivity information of its parent. The force-directed algorithm can be applied to this sequence of graphs, from small to large, each time using the layout of the smaller graph as the start layout for the larger graph. Combining the multilevel algorithm and the force approximation technique, algorithms based on the spring-electrical model can be used to layout graphs with millions of nodes and edges in seconds (Walshaw 2003; Hu 2005).

Spring/Stress Model

The spring model, also known as the stress model, assumes that there are springs connecting all pairs of nodes in the graph, with the ideal spring length equal to the graph theoretical distance between the nodes. The spring energy, also known as the stress, of this spring system is

$$\sum_{i \neq j} w_{ij} (\|x_i - x_j\| - d_{ij})^2, \quad (3)$$

where d_{ij} is the ideal distance between nodes i and j . The layout that minimizes the above stress energy is an optimal layout of the graph. Typically $w_{ij} = 1/d_{ij}^2$.

The spring model was proposed by Kamada and Kawai (1989) in graph drawing, although it dates back to multidimensional scaling (MDS) (Kruskal 1964; Kruskal and Seery 1980), and the term MDS is sometimes used to describe the embedding process based on the stress model.

There are several ways to minimize the spring energy (3). A force-directed algorithm could be applied, where the force exerted on node i from all other nodes j ($j \neq i$) is

$$F(i, j) = -w_{ij} (\|x_i - x_j\| - d_{ij}) \frac{x_i - x_j}{\|x_i - x_j\|}. \quad (4)$$

In recent years the stress majorization technique (Gansner et al. 2004) became a preferred way to minimize the stress model due to its robustness.

In the stress majorization process, systems of linear equations are repeatedly solved:

$$L_w X = L_{w,d} Y \quad (5)$$

here X and Y are matrices of dimension $n \times 2$, and the weighted Laplacian matrix L_w is positive semi-definite and has elements

$$(L_w)_{ij} = \begin{cases} \sum_{l \neq i} w_{il}, & i = j \\ -w_{ij}, & i \neq j \end{cases} \quad (6)$$

and the right-hand-side $L_{w,d} Y$ has elements

$$(L_{w,d} Y)_i = \sum_{l \neq i} w_{il} d_{il} (y_i - y_l) / \|y_i - y_l\|. \quad (7)$$

Here matrix Y is the current best guess of the optimal layout in 2D. The solution X serves as Y in the next iteration. A good initial layout Y is often helpful to achieve a good final layout.

On large graphs, the stress model (3) is not scalable. Formulating the model requires computing the all-pairs shortest path distances and a quadratic amount of memory to store the distance matrix. Furthermore the solution process has a computational complexity at least quadratic in the number of nodes. In recent years there have been attempts (Brandes and Pich 2007; Gansner et al. 2013; Ortmann et al. 2016) at developing more scalable ways of drawing graphs that still satisfy the user-specified edge length as much as possible, without having to carry out the all-pairs shortest path distances.

Directed Graph Drawing Algorithms

Directed graphs express relations that have a source and a target, e.g., senders and recipients of messages or hierarchical relationships in an organization. Using arrowheads to indicate directions, any undirected graph layout algorithm can be used for visualizing directed graphs. Yet, often the edge directions carry important information that should directly influence the layout geometry. In this section we discuss layout algorithms that make use of edge directions.

Layered Graph Layout

Layered graph layout is particularly useful for hierarchical graphs, which have no or only few cyclic relationships. For such graphs it is natural to ask for a drawing, in which all or most edges point into the same direction, e.g., upward. A classic algorithmic framework for upward graph layouts is layered graph drawing, often also known as Sugiyama layout named after the fundamental paper by Sugiyama et al. (1981). Each node is assigned to a layer, and for each edge it is required that the layer of the source node is below the layer of the target node. For a detailed survey of layered graph layout methods, see Healy and Nikolov (2014).

The process of computing layered graph layouts consists of a series of four steps. Each step influences the input to the subsequent steps and thus also the achievable layout quality.

1. **Cycle removal.** If the graph contains directed cycles, this step determines a small set of so-called *feedback edges* whose reversal yields an *acyclic* graph. The subsequent steps will draw this acyclic graph and in the end of the process the original edge directions are restored. Finding a minimum set of edges to reverse is equivalent to finding a so-called *minimum feedback arc set*, a classical NP-hard problem (Karp 1972). Several heuristics and approximation algorithms can be used to find small feedback edge sets in practice (Gansner et al. 1993; Eades et al. 1993).
2. **Layer assignment.** This step assigns each node to a horizontal layer, which can be thought of as an integer y-coordinate, such

that all edges point upward. Multiple nodes can be assigned to the same layer. Typical goals in this phase are to find compact layer assignments that use few layers and distribute nodes evenly to the layers, while ensuring that all edges point from a lower layer to a higher layer. After the layer assignment, some edges may span several layers. Such long edges are usually subdivided by dummy nodes to guarantee that for the next step, all edges connect nodes in neighboring layers. In order to keep edges short, a natural goal is to minimize the number of dummy nodes needed. Suitable layer assignments can be computed efficiently for most optimization goals, e.g., minimizing the number of layers based on topological sorting of the graph. For compact layouts with few dummy nodes and bounds on the number of nodes per layer, the ILP-based algorithm of Gansner et al. (1993) provides good results.

3. **Crossing minimization.** The crossing minimization step needs to determine the node order in each layer such that the number of edge crossings among the edges between two neighboring layers is minimized. Crossing minimization between two layers is again an NP-hard optimization problem (Eades and Wormald 1994). Several heuristics and (non-polynomial) exact algorithms for minimizing crossings between neighboring layers are known and evaluated in the experimental study by Jnger and Mutzel (1997). For computing node orders in all layers, usually a layer-by-layer sweep method is applied that cycles through all layers and optimizes node orders in neighboring layers, but global multilevel optimization methods exist as well (Chimani et al. 2011).
4. **Edge routing.** Once layers and node orders in each layer are fixed, it remains to assign x-coordinates to all nodes (including dummy nodes) that respect the node orders and optimize the resulting edge shapes. A typical optimization goal is to draw edges as vertical as possible with few bends for long edges (or with at most two bends with a strictly vertical middle segment) while

maintaining a minimum node spacing in each layer and avoiding node-edge overlaps. Algorithms for coordinate assignment either use mathematical programming (Sugiyama et al. 1981; Gansner et al. 1993) or a linear-time sweep method (Brandes and Kpf 2002).

Layouts for Specific Graph Classes

Previous sections covered visualization algorithms for general undirected and directed graphs. There is a variety of tailored algorithms for more specific graph classes, including prominent examples such as trees and planar graphs. These algorithms exploit structural properties when optimizing certain layout aesthetics. It is beyond the scope of this chapter to cover such specific graph classes and their visualization algorithms. Rather we refer to Rusu (2013) for a survey on tree layout and to Duncan and Goodrich (2013) and Vismara (2013) for surveys on planar graph layout. Further general books on graph drawing algorithms (Di Battista et al. 1999; Tamassia 2013) contain chapters on specific layout algorithms.

Examples of Application: Graph Visualization Software

There are many software packages and frameworks for visualizing and drawing graphs. A non-exhaustive list of noncommercial ones is given below. We divide the list into two parts. The following are packages that can handle relatively large graphs:

- Cytoscape is a Java-based software platform particularly popular with the biology community for visualizing molecular interaction networks and biological pathways.
- Gephi is a Java-based network analysis and visualization software package which is able to handle static and dynamic graphs. It is supported by a consortium of French organizations.
- Graphviz is one of the oldest open-source graph layout and rendering engines, developed at AT&T Labs. It is written in C and

C++ and hosts layout algorithms for both undirected (multilevel spring-electrical and stress models) and directed graphs (layered layout), as well as various graph theory algorithms. Graphviz is incorporated in Doxygen and available via R and Python.

- OGDF is a C++ class library for automatic layout of diagrams. Developed and supported by German and Australian researchers, it contains spring-electrical algorithms with fast multipole force approximation, as well as layered, orthogonal, and planar layout algorithms.
- Tulip is a C++ framework originating from the University of Bordeaux I for developing interactive information visualization applications. One of the goals of Tulip is to facilitate the reuse of components; it integrates OGDF graph layout algorithms as plugins.

G

The following are a few other free software each with its own unique merit, but not designed to work on very large graphs. For example,

- Dunnart is a C++ diagram editor developed at Monash University, Australia. Its unique feature is the ability to layout diagrams with constraints.
- D3.js is a popular JavaScript library for manipulating web documents based on data. It contains spring-electrical model-based layout modules solved by a force-directed algorithm. Since D3 works with SVG, it cannot scale beyond a few thousand graphical objects. However, a WebGL-based JavaScript library Viva-Graph could be used for larger graph visualization in the browser.

Future Directions for Research

Since the 1980s, a great deal of progress has been made in laying out graphs. The key enabling techniques are fast force approximations, the multilevel approach, and techniques for the efficient solution of the stress models. In addition, progress in the speed of GPU and graphics

library also made it possible to display graphs with millions of nodes and edges. Furthermore, there has also been progress in abstracting the visual complexity of large graphs, for example, by grouping similar nodes together, and representing certain subgraphs such as cliques as a motif. But as graphs become increasingly large, complex, and time-dependent, there are a number of challenges to be addressed.

The Increasing Size of the Graphs

The size of graphs is increasing exponentially over the years (Davis and Hu 2011). Social networks are one area where we have graphs of unprecedented size. As of late 2017, Facebook, for example, has over 2.07 billion monthly active users, while Twitter has over 330 million. Other data sources may be smaller but just as complex. For instance, Wikipedia currently has 5.5 million interlinked articles, while Amazon offers around 400 million items, with recommendations connecting each item to other like items. All these pale in comparison when we consider that there are 100 billion interconnected neurons in a typical human brain and trillions of websites on the Internet. Each of these graphs evolves over time. Furthermore, graphs like these tend to exhibit the small-world characteristic, where it is possible to reach any node from any other in a small number of hops. All these features present challenges to existing algorithms (Leskovec et al. 2009).

The unique features of these networks call for rethinking of the algorithms and visual encoding approaches. The large size of some networks means that finding a layout for such a graph can take hours even with the fastest algorithms available. There have been works on using GPUs and multiple CPUs, e.g., (Ingram et al. 2009), to speed up the computation by a factor of 10–100.

Even though state-of-the-art graph layout algorithms can handle graphs with many millions of nodes and billions of edges, with so many nodes and edges, the traditional node-link diagram representation is at its limit. A typical computer screen only has a few million pixels,

and we are running out of pixels just to render every node.

One solution is to use a display with high resolution, including display walls, and various novel ways to manipulate such a display (e.g., gesture- or touch-based control). But even the largest possible display is unlikely to be sufficient for rendering some of the largest social networks. One estimate puts human eyes as having a resolution of just over 500 million pixels. Therefore even if we can make a display with higher resolution, our eyes can only make partial use of such a display at any given moment.

Since the purpose of visualization is to help us to understand the structures and anomalies in the data, for very large graphs, it is likely that we need algorithms to discover structures and anomalies first (Akoglu et al. 2010) and display these in a less cluttered form, but allowing the human operator to drill down to details when needed.

Node-link diagram representation, while most common, may not be the most user-friendly to the general public, nor is it the most pixel-efficient. Other representations, such as a combination of node-link diagrams and matrices (Henry et al. 2007), have been proposed.

Large complex networks call for fast and interactive visualization systems to navigate around the massive amount of information. A number of helpful techniques for exploring graphs interactively, such as link sliding (Moscovich et al. 2009), have been proposed. Further research in this area, particularly at a scale that helps to make sense out of networks of billions of nodes and edges, are essential in helping us to understand large network data.

Finally, the stress model is currently the best algorithm for drawing graphs with predefined edge length. Improving its high computation and memory complexity is likely to remain an active area of research as well.

Time-Varying and Complex Graphs

All the large and complex networks mentioned earlier are time-evolving. How to visualize such dynamic networks is an active area of research (Frishman and Tal 2007), both in terms of graph

layout (van Ham and Wattenberg 2008), and in displaying such graphs. For example, time-varying graph can be displayed as an animation or as small multiples. Researchers have been studying Archambault and Purchase (2016) whether to use one form or the other, when measured in terms of preservation of the users' mental maps, and in improving comprehension and information recall. Dynamic network visualization will likely continue to be an area of strong interest.

Visualizing multivariate graphs is another challenging area. In a multivariate graph, each node and edge may be associated with several attributes. For example, in a social network, each node is a person with many possible attributes. Each edge represents a friendship which could entail multiple attributes as well, such as the type of friendship (e.g., school, work, church) and the length and strength of the friendship. Displaying these information in a way that helps our understanding of all these complex attributes is a challenge. It requires careful consideration of both visual design and interaction techniques (Wattenberg 2006; Kerren et al. 2014).

Cross-References

- ▶ [Big Data Visualization Tools](#)
- ▶ [Graph Exploration and Search](#)
- ▶ [Visualization](#)
- ▶ [Visualization Techniques](#)

References

- Akoglu L, McGlohon M, Faloutsos C (2010) Oddball: Spotting anomalies in weighted graphs. In: Proceedings of the 14th Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD 2010)
- Archambault D, Purchase HC (2016) Can animation support the visualisation of dynamic graphs? *Inf Sci* 330(C):495–509
- Brandes U, Kpf B (2002) Fast and simple horizontal coordinate assignment. In: Mutzel P, Jnger M, Leipert S (eds) Graph drawing (GD'01). LNCS, vol 2265. Springer, Berlin/Heidelberg, pp 31–44
- Brandes U, Pich C (2007) Eigensolver methods for progressive multidimensional scaling of large data. In: Proceeding of the 14th international symposium graph drawing (GD'06). LNCS, vol 4372, pp 42–53
- Chimani M, Hungerlnder P, Jnger M, Mutzel P (2011) An SDP approach to multi-level crossing minimization. In: Algorithm engineering and experiments (ALENEX'11), pp 116–126
- Davis TA, Hu Y (2011) University of Florida sparse matrix collection. *ACM Trans Math Softw* 38:1–18
- Di Battista G, Eades P, Tamassia R, Tollis IG (1999) Algorithms for the visualization of graphs. Prentice-Hall, Upper Saddle River
- Duncan CA, Goodrich MT (2013) Planar orthogonal and polyline drawing algorithms, chap 7. In: Tamassia R (ed) Handbook of graph drawing and visualization. CRC Press, Hoboken, pp 223–246
- Eades P (1984) A heuristic for graph drawing. *Congressus Numerantium* 42:149–160
- Eades P, Wormald NC (1994) Edge crossings in drawings of bipartite graphs. *Algorithmica* 11:379–403
- Eades P, Lin X, Smyth WF (1993) A fast and effective heuristic for the feedback arc set problem. *Inf Process Lett* 47(6):319–323
- Frishman Y, Tal A (2007) Online dynamic graph drawing. In: Proceeding of eurographics/IEEE VGTC symposium on visualization (EuroVis), pp 75–82
- Fruchterman TMJ, Reingold EM (1991) Graph drawing by force directed placement. *Softw Prac Exp* 21:1129–1164
- Gansner ER, Koutsofios E, North SC, Vo KP (1993) A technique for drawing directed graphs. *IEEE Trans Softw Eng* 19(3):214–230
- Gansner ER, Koren Y, North SC (2004) Graph drawing by stress majorization. In: Proceeding of the 12th international symposium on graph drawing (GD'04). LNCS, vol 3383. Springer, pp 239–250
- Gansner ER, Hu Y, North SC (2013) A maxent-stress model for graph layout. *IEEE Trans Vis Comput Graph* 19(6):927–940
- Hachul S, Jnger M (2004) Drawing large graphs with a potential field based multilevel algorithm. In: Proceeding of the 12th international symposium graph drawing (GD'04). LNCS, vol 3383. Springer, pp 285–295
- Healy P, Nikolov NS (2014) Hierarchical drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization, chap 13. CRC Press, Boca Raton, pp 409–454
- Henry N, Fekete JD, McGuffin MJ (2007) Nodetrix: a hybrid visualization of social networks. *IEEE Trans Vis Comput Graph* 13:1302–1309
- Hu Y (2005) Efficient and high quality force-directed graph drawing. *Math J* 10:37–71
- Ingram S, Munzner T, Olano M (2009) Glimmer: multi-level MDS on the GPU. *IEEE Trans Vis Comput Graph* 15:249–261
- Jnger M, Mutzel P (1997) 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *J Graph Algorithms Appl* 1(1):1–25

- Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. *Inform Process Lett* 31: 7–15
- Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, Bohlinger JD (eds) Complexity of computer computations, pp 85–103.
- Kerren A, Purchase H, Ward MO (eds) (2014) Multivariate network visualization: Dagstuhl seminar # 13201, Dagstuhl Castle, 12–17 May 2013. Revised discussions, Lecture notes in computer science, vol 8380, Springer, Cham
- Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29:1–27
- Kruskal JB, Seery JB (1980) Designing network diagrams. In: Proceedings of the first general conference on social graphics, U.S. Department of the Census, Washington, DC, pp 22–50, Bell laboratories technical report no. 49
- van Ham F, Wattenberg M (2008) Centrality based visualization of small world graphs. *Comput Graph Forum* 27(3):975–982
- von Landesberger T, Kuijper A, Schreck T, Kohlhammer J, van Wijk JJ, Fekete JD, Fellner DW (2011) Visual analysis of large graphs: state-of-the-art and future research challenges. *Comput Graph Forum* 30(6): 1719–1749
- Leskovec J, Lang K, Dasgupta A, Mahoney M (2009) Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. *Internet Math* 6:29–123
- Moscovich T, Chevalier F, Henry N, Pietriga E, Fekete J (2009) Topology-aware navigation in large networks. In: CHI '09: Proceedings of the 27th international conference on human factors in computing systems. ACM, New York, pp 2319–2328
- Ortmann M, Klimenta M, Brandes U (2016) A sparse stress model. In: Graph drawing and network visualization – 24th international symposium, GD 2016, Athens, revised selected papers, pp 18–32
- Purchase HC (1997) Which aesthetic has the greatest effect on human understanding? In: Proceeding of the 5th international symposium graph drawing (GD'97). LNCS. Springer, pp 248–261
- Quigley A (2001) Large scale relational information visualization, clustering, and abstraction. PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle
- Rusu A (2013) Tree drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization chap 5. CRC Press, Boca Raton, pp 155–192
- Sugiyama K, Tagawa S, Toda M (1981) Methods for visual understanding of hierarchical systems. *IEEE Trans Syst Man Cybern SMC-11(2)*:109–125
- Tamassia R (2013) Handbook of graph drawing and visualization. Chapman & Hall/CRC, Boca Raton
- Tunkelang D (1999) A numerical optimization approach to general graph drawing. PhD thesis, Carnegie Mellon University
- Vismara L (2013) Planar straight-line drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization chap 6. CRC Press, Boca Raton, pp 193–222
- Walshaw C (2003) A multilevel algorithm for force-directed graph drawing. *J Graph Algorithms Appl* 7:253–285
- Wattenberg M (2006) Visual exploration of multivariate graphs. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI'06). ACM, New York, pp 811–819

Green Big Data

► [Energy Implications of Big Data](#)

H

Hadoop

Renata Ghislotti Duarte de Souza Granha Bosch, Chicago, USA

Synonyms

Hadoop ecosystem; Hadoop software; HDFS; MapReduce

Definition

Apache Hadoop is an open-source platform for storage and efficient processing of large datasets on a cluster of computers. The framework provides fault tolerance, high availability, and scalability, being able to process petabytes of data. Its principal components are MapReduce and HDFS.

Overview

Introduction

Apache Hadoop is a distributed framework used to tackle Big Data. It is a software platform in a master/worker architecture with three main components: HDFS, YARN, and MapReduce. The HDFS (Hadoop Distributed File System) is an abstraction layer responsible for the storage of data. MapReduce is the data processing framework

designed specifically to scale and run distributed. YARN (Yet Another Resource Negotiator) is a management platform responsible for handling resources in the cluster. Hadoop's open-source software was written in Java and distributed under Apache license 2.0.

The Hadoop framework can be implemented on a cluster of relatively inexpensive computers since it already works with the premise that machines fail and the system should be prepared to handle that. Besides fault tolerance, it provides high availability and scalability while hiding most distributed computing details from the user. These qualities made Hadoop a well-adopted solution in the industry (White 2015).

It is important to note that Hadoop is not a replacement for relational databases. The reason for this is that Hadoop was conceived for batch processing, where higher latency is acceptable. Data in Hadoop is not indexed; therefore seeking specific patterns can be a time-consuming task. Nevertheless, real-time analytics is a possibility in Hadoop together with some other Apache components such as HBase, Kafka, and Spark. Apache HBase is a no-sql column table database that runs on top of HDFS. Apache Kafka is a stream processing platform that enables handling real-time data. Apache Spark offers a component, Spark Streaming that works together with Kafka for building streaming applications.

Currently, Hadoop is a term used to express an entire toolset of softwares that provide services related to Data Science and Big Data used mostly alongside or on top of Hadoop software

itself. Each tool has its own application and advantages. The Hadoop ecosystem comprises softwares for data processing such as Apache Hive, Apache Spark and Apache Drill, Apache Oozie for jobs orchestration, Apache Mahout and Apache Spark's MLlib for machine learning applications, and many others.

Hadoop History

The Apache Hadoop project started as a part of the Apache Nutch, a scalable open-source web crawler project. Nutch was created in 2002 as a component of the Apache Lucene initiative. Doug Cutting was the creator of Apache Lucene and started Nutch trying to build a web crawler to democratize search engine algorithms. He soon realized that in order to do that he would need to scale Nutch to the complexity of the Internet. Hadoop was the answer to this quest.

Doug Cutting conceived Hadoop inspired by Google's 2003 and 2004 papers on Google's distributed file system (GFS) (Ghemawat et al. 2003) and distributed programming paradigm MapReduce (Dean and Ghemawat 2008). The GFS paper described the foundations of distributed storage for data intensive applications used by the company. It explained how to deploy such a platform on commodity hardware while providing fault tolerance and scalable performance. The MapReduce paper introduced a straightforward paradigm to do distributed programming using the GFS architecture. The programmer would be responsible for providing a Map and a Reduce function to process the data, while all resource allocation and application management were handled by the platform. These two components put together made possible to scale Nutch, and Doug soon realized that Hadoop scalability had great applicability to other areas.

In 2006, Hadoop emerged as an independent Apache project, and around that same time, Doug Cutting joined Yahoo!. The company saw the tool potential and created a dedicated team to develop it. The name Hadoop was chosen by Doug inspired in his kid's toy, a yellow stuffed elephant. By 2008, Hadoop was already a top-level Apache project, and by 2010 a whole ecosystem of tools

was created around Hadoop, such as Apache HBase, Hive, and Pig.

The first version of Hadoop, known as Hadoop 1, consisted of two main components: MapReduce and HDFS. In this initial version, MapReduce was responsible for managing job resources. Around 2012, YARN was introduced in Hadoop 2, making it much more polyvalent and open to new processing engines such as Apache Spark. Currently, the use of MapReduce paradigm for data processing has declined, being replaced by higher-level APIs such as Hive or by more dynamic processing tools like Spark. HDFS remains a widely adopted distributed file system.

Hadoop Components

HDFS

The Hadoop Distributed File System is an abstraction layer between the regular operating system file system and a data processing framework such as MapReduce, Spark, or HBase. It was written in Java in a *shared nothing master-worker architecture*. In Hadoop, computers are nodes in a cluster, and they exchange information through the network (usually using a TCP/IP socket communication). The master node orchestrates the cluster and the workers do the computation. HDFS was thought to handle big datasets providing high availability and fault tolerance.

The master, *NameNode*, is responsible for storing all files and directories metadata such as file access permission and file locations. The file locations are not stored persistently on disk in the NameNode; they stay in memory for quicker access. Note that this limits the amount of data that is possible to store in a Hadoop system: the amount of files cannot be larger than what the NameNode can store in memory. This is also one of the reasons why HDFS deals better with larger files than a large number of small files; small files have greater chances of overloading NameNode's memory. Usually, each file block or directory takes about 150 bytes of memory.

The *Secondary NameNode* exists to help the NameNode, by copying the NameNode file metadata. Note that these copies are not guaranteed

to be updated, so in case of failure, the Secondary NameNode is not a hot standby to the NameNode. With Apache Zookeeper, however, it is possible to arrange the Secondary NameNode to be a hot standby. It is also possible to have multiple NameNodes taking care of multiple clusters together with the so-called HDFS Federation, a concept introduced in Hadoop 2.

The worker DataNodes are responsible for storing and processing data. When storing a file in HDFS, each file is divided into blocks, and each block is replicated and stored separately in different DataNodes. The default replication factor is three and the default block size is 128 MB. The blocks are actually located in the regular OS file system, but one accesses it through HDFS's API.

Replicating data is important for the failover mechanism, for high availability, and for load balancing. Node failures are detected by the Hadoop system with a heartbeat type of signaling. Each DataNode sends heartbeat signals to the NameNode every 3 s. If a DataNode fails, the other two replicas can be used to ensure high availability. In that case, when the NameNode stops receiving the heartbeat signal, it will copy the lost DataNode data into another DataNode to maintain the default replication factor. To replicate the lost data, the NameNode uses one of the other two copies of the data available in HDFS. Now, assume that no DataNode is failing but many of them are overloaded with processing jobs. Having three data replicas helps ensuring that processing power will be available to new jobs, and processing is distributed more or less equally among nodes. Furthermore, it helps ensure the **data locality** principle of sending the job to the data (and not otherwise) so as to not overload the network traffic.

There are three main ways of interacting with HDFS: a command-line interface, a Java API, and a web interface with Apache Hue. The command-line interface is similar to POSIX, presenting commands such as *mkdir*, *ls*, and *cat*.

When writing a file into HDFS with the command-line interface, a file can be inserted in the system with the following command:

```
$ hadoop hdfs -put my_file.txt
```

Under the hood, the client is sending a message to the NameNode asking permission to write a file. If the client's permission is granted, the NameNode answers with DataNode's locations to write the blocks. The file *my_file.txt* is then divided into blocks, and each block is copied to one different DataNode. Note that the file does not pass through the NameNode itself which would cause a bottleneck in the system. Instead, the client sends the block to the closest DataNode, and this DataNode by its turn sends the block to the second closest DataNode and so on.

As mentioned, Hadoop replicates the block three times. This, as it is with most Hadoop parameters, can be configured by the cluster administrator. Two data blocks are copied to different machines on a same rack, trying to reduce as much as possible the network traffic. This feature is called *rack awareness*. A third copy is placed on a different rack to ensure data availability in case of a rack failure. To ensure data integrity, a checksum of the data is calculated every time it is read or written. All of these actions are handled by the system and are transparent to the user.

YARN

YARN was introduced in Hadoop version 2 to separate the resource management function from the processing layer of Hadoop, thus providing a broader framework. Because of YARN, Hadoop was able to support other processing models such as iterative programming or graph processing.

The platform contains a Resource Manager to distribute the resources of the cluster among all applications. YARN provides each application with an ApplicationMaster, which talks with the Resource Manager to negotiate resources. Another important entity is the NodeManager, which is responsible for launching and monitoring each application.

MapReduce

MapReduce is a programming paradigm written in Java to run on top of the HDFS, inspired by Google's MapReduce paper (Dean and Ghemawat 2008). It was mainly envisioned for batch processing on embarrassingly parallel problems where the two functions Map and

Reduce process data in a resilient and distributed manner. Many real-world computer science problems have already been modeled into the MapReduce paradigm (Rajaraman and Ullman 2011).

Suppose that a group of people want to count how many books are there in a library. They decide to divide the work, and each person is responsible for counting books in certain shelves. The group finally gets together and sum up each individual count to obtain an overall amount. This same principle is used in the MapReduce framework. The Map phase applies operations in separated chunks of the dataset, and the Reduce phase can be used to combine results appropriately. The Map and Reduce functions come from functional programming and are easily parallelizable. In this paradigm, the user is responsible for specifying a Map and a Reduce function and the input and output location to read and write data. All distributed programming details such as input partitioning, job failures, and resource scheduling are handled by the system so that the user can focus on processing data. This framework allows user with no distributed programming experience to do distributed computing.

MapReduce operates with key/value pairs. The input of a Map function should be in the key/value format, such as (K1, V1), and the output as a list of key/values, list(K2, V2). The Reduce function should receive as input a tuple (k2, list(v2)) and return as output a list of key/value pairs, for example, list(K3, V3). Note that the input of the Reduce function must match the output of the Map function, being an aggregation on values by key. Other than that, the key/value pairs can be of any kind.

Given a Map and a Reduce function, the system will partition the dataset in blocks, as mentioned before, and create several Map and Reduce tasks to run on the data. The client application will talk to an agent called *Resource Manager* from the YARN component, to ask for resources to run the application. The Resource Manager will try to allocate a resource container in a DataNode close to the data, based on the data locality principle. It will first try to allocate re-

sources for the task in the same DataNode where data is located. If it can't, it will try a node on the same rack, and if that is not available as well, it will go for any other node in the cluster. It is important to highlight that the Resource Manager will execute the Map and Reduce functions in a separate JVM, so in case of any failures, the rest of the system should not be affected.

There will be one Map task per block of data. The tracking of tasks is made by YARN through heartbeat signals sent from the tasks to the Resource Manager. The system executes the tasks in parallel, and if any of the tasks fail, it simply re-executes that single task, and not the entire application program. The intermediate results of Map tasks are written persistently to disk, and not into HDFS. This is done because it would be too expensive to write all intermediate data into HDFS, and in case the data is lost, it is less expensive to just run the Map task again. Intermediate result is written to disk and it is sorted in a phase known as *Shuffle*. All data with the same key is grouped together and fed to a single Reduce task using a hash function. One Reduce task might receive input from many or all Map tasks. The Reduce task might be in the same or in another DataNode in the system; this is determined by the Resource Manager. The Reduce phase doesn't wait for all Map tasks to finish to start shuffling data, but it has to wait all Map tasks to finish to complete the shuffle. The shuffling phase is usually the most time-consuming part of a MapReduce job, since possibly all data has to be ordered. To help optimizing this phase, the MapReduce framework provides *Combiners* to aggregate and reduce data before sending it to the Reduce function. Combiners can be applied in some MapReduce jobs. Finally, once Reduce is done, its output is written to HDFS.

Several packages and libraries were created to support the development of MapReduce applications. The *Hadoop Streaming* component provides an API for users to write Map and Reduce functions in languages other than Java, like Python, Ruby, or C. A library; MRUnit was created with JUnit to support testing in MapReduce.

The Apache Oozie project can be useful for the development of applications that use several Map and Reduce tasks sequentially, since it works as a job orchestrator and scheduler.

MapReduce Limitations

The MapReduce paradigm facilitates distributed programming, with the idea of breaking down a dataset and applying in parallel the same function to all portions of data. There are some drawbacks nonetheless. MapReduce is mainly applicable to embarrassingly parallel programs, where communication between processes is minimal and the algorithm is easily parallelized. When willing to do iterative analysis or communication intensive applications, users have to concatenate several Map and Reduce functions sequentially, which may greatly increasing the running time. The reason for this is that each time a Map function sends data to a Reduce function, it writes data on disk. Therefore, the overhead of writing and reading from disk makes iterative analysis on MapReduce less attractive. Apache Spark was developed to overcome this issue. It processes data in memory, by applying functions sequentially without writing to disk. It creates a directed acyclic graph of tasks, where the state of the transformations applied to the data are saved for fault tolerance. This mechanism together with Spark's simple API made it a great resource for data processing. In 2016, Hadoop's creator Doug Cutting declared that Spark is the evolution of the MapReduce framework (Cutting 2016).

Examples of Application

Several real-world applications can be modeled as MapReduce problems. Recommender systems, a technique used to advertise products to users, have been extensively expressed in terms of MapReduce (Rajaraman and Ullman 2011). The frequent itemset problem, k-mean clustering, and many other machine learning problems were modeled in the MapReduce paradigm in the book *Mining of Massive Datasets* (Rajaraman and Ullman 2011).

Google has publicly announced in 2004 that it used MapReduce on its search engine with more than 20 TB of data, as well as in clustering applications for Google News, construction of graph application, and several others (Rajaraman and Ullman 2008).

Hadoop has also been used extensively by Yahoo!. In 2008, Yahoo! announced that they used a 10,000 core cluster to index its search engine, and in 2009 they could sort 1 terabyte of data in 62 s (White 2015). By 2011, Yahoo! had a 42,000 node Hadoop cluster with petabytes of storage (Harris 2013).

H

References

- Cutting D (2016) <https://www.youtube.com/watch?v=Phjif5vAhM>. Accessed 20 Oct 2017
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1): 107–113. <https://doi.org/10.1145/1327452.1327492>
- Ghemawat S, Gobioff H, Leung S-T (2003) The Google file system. SIGOPS Oper Syst Rev 37(5):29–43. <https://doi.org/10.1145/1165389.945450>
- Harris D (2013) <https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/>. Accessed 20 Oct 2017
- Rajaraman A, Ullman JD (2011) Mining of massive datasets. Cambridge University Press, New York
- White T (2015) Hadoop: the definitive guide, 4th edn. O'Reilly Media, Hadoop

Hadoop Ecosystem

► Hadoop

Hadoop EDW Integration

► Hybrid Systems Based on Traditional Database Extensions

Hadoop Software

► Hadoop

Hadoop-Based RDF Query Processors

- ▶ Framework-Based Scale-Out RDF Systems

Hardware Considerations for Big Data

- ▶ Computer Architecture for Big Data

Hardware Reliability Requirements

Wenhai Li

Advanced Computing Research Centre, Data to Decisions Cooperative Research Centre, University of South Australia, Adelaide, Australia

Introduction

The development of big data applications has brought a great amount of opportunities yet challenges to both the IT industry and academia. Supporting platforms and architectures, such as Clouds, supercomputing centers and data centers, must develop corresponding technologies to meet the “4V” requirements of big data applications, which are veracity, velocity, volume, and variety. Serving as the fundamental layer of these platforms and architectures, the hardware, especially storage hardware, are of great importance to the big data applications, where the reliability of storage units not only have significantly influence the reliability of the data, but also have significant impact on the “4V” requirements of the big data applications. In this article, we discuss the hardware reliability requirements, specifically, storage hardware reliability requirements, for current big data platforms and architectures.

Big Data Storage Architecture

A typical big data application contains several terabytes to several petabytes of input/output data and intermediate data in either structured or unstructured formats. It is not surprising that the storage architecture plays the dominating role in the reliability of the data and the functionality of the applications. Traditionally, the data is stored in local storage architectures such as hard disk racks and connected via storage networks. Nowadays, more and more companies are shifting their data into large-scale data centers or public clouds to prevent huge investment into the storage hardware, maintenance of data/redundancy, long-term energy cost, as well as the risk of losing the data. In the foreseeable future, local storage and Cloud/data center storage will coexist due to security and performance reasons. However, instead of network-attached storage (NAS), the local storage is nowadays gradually adopting more advanced technologies, such as private cloud or distributed storage system, to meet the extremely high accessing and processing requirements of the big data applications. In terms of local storage architecture leveraging the latest technologies and the public cloud and large-scale data center, these two storage structure share very similar design, and hence share the same group of factors that affect reliability.

Factors that Affect Reliability

Existing research and industrial reports have shown several factors that could affect the reliability of the hardware for data storage architectures:

- Uncontrollable natural or manmade disasters (Dabrowski 2009; Wood et al. 2010): Natural disasters such as floods, hurricanes, and earthquakes or manmade disasters such as infrastructure failure, disastrous IT bugs, or failed hardware installations could cause serious damage to the IT facilities and hence the loss of data.
- Environmental factors (Gersbach 2012; Loken et al. 2010): Environmental factors such as

- temperature, static electricity, and humidity surrounding the IT facilities could impact the reliability of the hardware, where harsh environment could significantly reduce the durability and raise the failure rates of storage components.
- Storage media (Mielke et al. 2008; Pinheiro et al. 2007; Schroeder and Gibson 2007; Zaman 2016; Furrer et al. 2017): Different storage medias have different reliability characteristics. In modern data storage architectures, commonly used storage media includes hard disk drive (HDD), solid-state drive (SSD), and magnetic tape. These storage medias could coexist in a single data storage architecture for storing different types of data according to the reliability requirement, performance requirement, and cost-effective requirement. Details of the storage medias will be described in section “[Reliability of Storage Medias](#).”
 - Media controller (Zaman 2016; Patterson et al. 1988; Taylor 2014): Above the very bottom storage media, there could be another layer of hardware named media controller that is designed to improve the reliability and performance of the storage facilities. For example, multiple hard disk drives could be grouped by a RAID controller to build a RAID array, in which different configurations can be applied to improve the throughput performance and the reliability of the data, respectively. For solid-state drives, a media controller (or SSD controller) is a part of the SSD firmware, in which error correction is conducted using error correction coding technology and wear leveling is conducted to make sure all the flash cells experience the same amount of erase/write cycles so that the chip ages evenly. Different error correction code and wear leveling algorithms can be applied to the SSD controller by reprogramming it to maximize the reliability of the drive.
 - Processing hardware (Dabrowski 2009; Pinheiro et al. 2007; Schroeder et al. 2016): Unexpected non-storage hardware failure, such as motherboard failure or RAM failure, could also cause loss of data in the data storage architecture, especially when the data writing process is in progress. Compare to the reliability of storage media, these hardware parts cause less impact to the reliability of the data (Pinheiro et al. 2007; Schroeder et al. 2016), and technics regarding this type of failure have been well developed for a fairly long period of time (Dabrowski 2009).
 - Software reliability mechanism: Different reliability assurance approaches are also applied in data management software and operating systems to make sure that hardware malfunction cause minimum impact on the application and the data. In this field, many technics have been developed for decades such as data replication (Li et al. 2015), erasure coding (Rashmi et al. 2014), checkpoint/snapshot(Dabrowski 2009), and recovery(Rashmi et al. 2013). Details of this topic are out of scope of this entry and will be covered in chapter “[▶ Data Replication and Encoding](#)” of the encyclopedia.

Reliability of Storage Medias

The reliability of storage medias is the fundamental factor for a big data storage architecture. Simply, storage medias with lower reliability could cause higher probability and frequency of data loss and cause lower data availability. Given large amounts of data stored, the impact on big data applications is significantly magnified. Even with software reliability mechanism applied to provide certain fault tolerance and data recovery ability to the system, the probability that the data being totally lost can still be significantly increased. Analyses have shown that most server failures in a data center are due to the storage media (Vishwanath and Nagappan 2010). As mentioned earlier, each storage media has its own features. These features include response time, total throughput, failure rate and failure pattern, etc. Therefore, big data storage architectures can apply a mixed storage solution by combining HDD, SSD, and magnetic tape cartridges to provide comprehensive data storage solutions to meet different requirements of different big data

applications. For example, SSD and HDD can be applied as tier 1 storage for newly generated data or data that is frequently accessed and require high access speed and low response time, and magnetic tape can be applied as tier 2 storage for data of very large volume, the so-called cold data that is not accessed very frequently as it takes hours to days to access. In current large-scale data centers, this solution is widely applied (Huang et al. 2012; Harris 2014). However, with the drop of storage cost by using HDD, another combination is getting more and more popular, where SSD serves as tier 1, HDD serves as tier 2, and magnetic tape is not used. In this section, we talk about the reliability features of HDD and SSD.

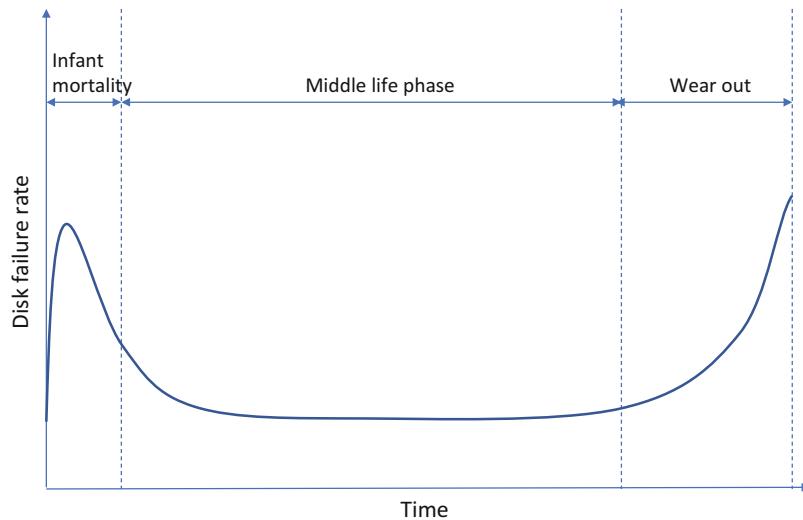
HDD

The reliability of HDDs has been investigated for decades in both academia and industry, where the disk failure rate is considered the main indicator. In a big data storage architecture, environmental factors are maintained to the optimum range, and unexpected dropping to the ground as normal home usage will not happen. However, the worn out of the internal physical parts could still cause a final failure of the drive. A typical hard disk's lifespan in a storage architecture is about 4 years. With the increase of the age of hard disks, existing studies have shown that the disk failure rate follows what is often called a "bathtub" curve, as shown in Fig. 1, where disk failure rate is higher in the disk's early life (infant mortality), drops during the first year, remains relatively constant for the remainder of the disk's useful lifespan (middle life phase), and rises again at the end of the disk's lifetime (wear-out) (Xin et al. 2005). An investigation conducted on a large disk drive population by Google has observed some results very consistent to this model (Pinheiro et al. 2007). For ease of representation, the International Disk Drive Equipment and Materials Association (IDEA) proposed a compromised presentation for disk failure rates, as shown in Fig. 2, where the lifespan of each disk is divided into different life stages with different failure rates, which are 0–3 months, 3–6 months, 6–12 months, and 1 year to the end of design lifespan (the wear-out phase of the disk is con-

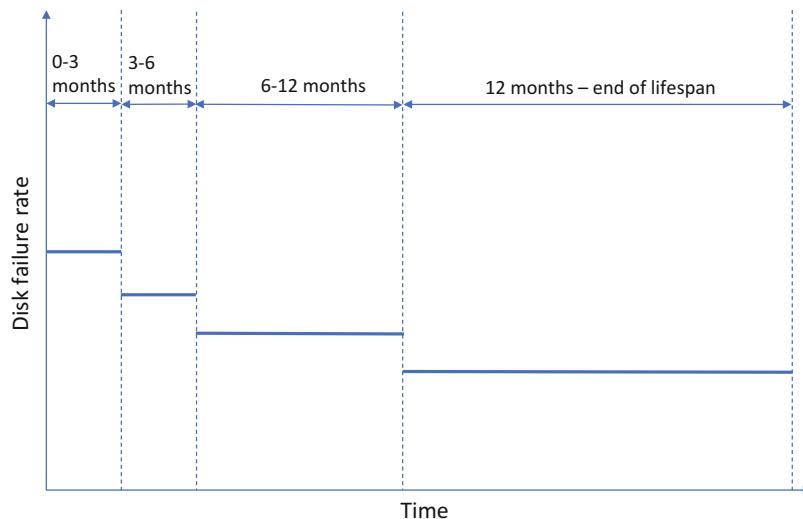
sidered out of useful life of the disk and hence not included in this model) (IDEA 1998). In (Pinheiro et al. 2007), it is reported that the average annual replacement rates of HDD is 2–9%. More recently in the data storage industry, hard disk reliability statistical reports are published regularly that review the disk failure rate of disks with different brands, capacities, performances, and prices, which tracks the reliability of the disks as end users (Klein 2017). The report has shown that the latest models of disks with the sizes ranging from 3 to 8 TB normally have an annual failure rate between 0.33% and 3.25%, where some extreme ones can be as low as 0.15% or as high as 31.58%.

SSD

The outstanding performance in I/O bandwidth and random IOPS (input/output operations per second) has made SSD highly desirable in current big data storage architectures. Current SSDs use NAND flash memory cells to retain data by trapping electrons inside. A process called tunneling is used to move electrons in and out of the cells, but the back-and-forth traffic erodes the physical structure of the cell, leading to breaches that can render it useless (Zaman 2016). Therefore, different from HDD, SSD's capability of storing data fails gradually, and its reliability is evaluated by raw bit error rate (RBER) or sometimes uncorrectable bit error rate (UBER) as it is easier to be observed. There are quite some existing research regarding to the reliability of SSD based on controlled lab tests under synthetic workloads, which predicts an exponential growth between RBER and number of program erase cycles (Mielke et al. 2008; CAI et al. 2012). However, in actual data center environments, it is observed that RBER grow linearly with program erase cycles, and same as HDD, it is also significantly affected by the age of the SSD (Schroeder et al. 2016). Compare to HDD, SSD has a lower replacement rate, which is 4–10% being replaced in a 4-year period compare to the HDD annual replacement rate of 2–9%. However, it also has a higher probability of generating uncorrectable bit, which means that while less likely to fail during usage, SSD is more likely of losing data.



Hardware Reliability Requirements, Fig. 1 Failure rate curve of hard disk drive



Hardware Reliability Requirements, Fig. 2 IDE MA failure rate curve of hard disk drive

Promising Future Trend

In general, with the big data application getting more and more popular nowadays, several trends can be expected in the near future in the data storage architectures.

1. It tends to share more and more of the same group of software and hardware technics on both local and Cloud/data center storage architectures. Software-wise, Cloud plat-

forms, distributed storage systems, and data processing systems are designed to be flexible and scalable regardless of the scale of underneath hardware, which makes the boundary of local and Cloud/data center storage systems less clear than before. Hardware-wise, both local and Cloud/data center storage architecture uses the so-called commodity hardware to maximize cost-effectiveness, which makes the difference even smaller.

2. In the hardware prospective, specifically, layered storage will become the trend. Firstly, as the unit price of SSD drops, it can be expected that SSD will be more widely used to maximize the I/O throughput, IOPS of the system to meet the high velocity requirement of big data applications. Secondly, HDD will still play a major role in the storage architecture due to its higher capacity, better durability, and less data loss rate to meet the high-volume and reliability requirement of the data. Thirdly, although magnetic tape still provides the lowest unit storage cost and largest storage capacity, the time-consuming accessing of data made it less desirable for many, if not all, big data applications and hence could be less applied in future data storage structures.

References

- CAI Y et al (2012) Error patterns in MLC NAND flash memory: measurement, characterization, and analysis. In: Design, automation & test in Europe conference & exhibition, Dresden
- Dabrowski C (2009) Reliability in grid computing systems. *Concurr Comput Pract Experience* 21(8): 927–959
- Furrer S, Lantz MA, Reininger P (2017) 201 Gb/in² recording areal density on sputtered magnetic tape. *IEEE Trans Magn* 99:1–1
- Gersbach T (2012) Technical article: Energy-efficient climate control “keep cool” in the data centre. RITTAL Pty, New South Wales
- Harris R (2014) Amazon’s Glacier secret: BDXL. [cited 2017]; Available from <https://storagemojo.com/2014/04/25/amazons-glacier-secret-bdxl/>
- Huang C et al (2012) Erasure coding in Windows Azure storage. In: USENIX annual technical conference, Boston
- IDEA (1998) R2–98: specification of hard disk drive reliability. IDEMA Standards
- Klein A (2017) Hard drive stats for Q2 2017. [cited 2017]; Available from <https://www.backblaze.com/blog/hard-drive-failure-stats-q2-2017/>
- Li W, Yang Y, Yuan D (2015) Ensuring cloud data reliability with minimum replication by proactive replica checking. *IEEE Trans Comput* 65:1–13
- Loken C et al (2010) SciNet: lessons learned from building a power-efficient top-20 system and data centre. *J Phys Conf Ser* 256(1):1–35
- Mielke N et al (2008) Bit error rate in NAND flash memories. In: IEEE international reliability physics symposium, Phoenix
- Patterson D, Gibson G, Katz R (1988) A case for redundant arrays of inexpensive disks (RAID). In: ACM SIGMOD international conference on the management of data, Chicago
- Pinheiro E, Weber W, Barroso LA (2007) Failure trends in a large disk drive population. In: USENIX conference on file and storage technologies, San Jose
- Rashmi KV et al (2013) A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster. In: USENIX HotStorage, San Jose
- Rashmi KV et al (2014) A hitchhiker’s guide to fast and efficient data reconstruction in erasure-coded data centers. In: SIGCOMM, Chicago
- Schroeder B, Gibson G (2007) Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? In: USENIX conference on file and storage technologies, San Jose
- Schroeder B, Lagisetty R, Merchant A (2016) Flash reliability in production the expected and the unexpected. In: USENIX conference on file and storage technologies, Santa Clara
- Taylor C (2014) SSD vs. HDD: performance and reliability. [cited 2017]; Available from <http://www.enterprisestorageforum.com/storage-hardware/ssd-vs-hdd-performance-and-reliability-1.html>
- Vishwanath KV, Nagappan N (2010) Characterizing cloud computing hardware reliability. In: ACM symposium on cloud computing, Indianapolis
- Wood T et al (2010) Disaster recovery as a cloud service: economic benefits & deployment challenges. In: HotCloud, Boston
- Xin Q, Schwarz TJE, Miller EL (2005) Disk infant mortality in large storage systems. In: IEEE international symposium on modeling, analysis, and simulation of computer and telecommunication systems, Atlanta
- Zaman R (2016) SSD vs HDD: which one is more reliable? [cited 2017]; Available from <https://therevisionist.org/reviews/ssd-vs-hdd-one-reliable/>

Hardware-Assisted Compression

K. Sayood and S. Balkir

Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

Definitions

Video compression: Compact representation of digital video.

Discrete cosine transform (DCT): An orthonormal transform used in many compression applications.

Arithmetic coding: An entropy coding technique that is particularly useful for alphabets with a skewed probability distribution.

LZ77: A dictionary based sequence compression algorithm which adaptively builds its dictionary through an optimal parsing of the “past” of the sequence.

Introduction

The information revolution has resulted in the ubiquity of the use of compression. As the explosion in the spread of information persists, the need for energy efficient compression is driving the development of more and more hardware-assisted compression, despite the increasing power of processors. The need is most where the resource constraints are most severe – where the constraints are relative to the application. Video compression deals with a huge amount of temporally sensitive information and thus is highly time-constrained. As video has become the dominant user of the available internet bandwidth, compression algorithms have become more and more complex in order to satisfy both the bandwidth constraints and the increasing quality requirements. The latter is due to significant improvements in display technology. This makes software implementations of video compression algorithms increasingly insufficient for the task at hand. At the other end of the spectrum, in terms of the volume of data, are wireless sensor networks which generally deal with significantly less data per node but have to process them under severe power constraints.

Because of the opportunistic nature of the deployment of hardware-assisted compression, it is difficult to provide a systematic description of the field; therefore, we will organize our discussion around the most popular applications – video compression, wireless sensor networks, and computer memory – and then take a somewhat meandering path through the other applications that do not fit into these categories. We will not

focus on the specific hardware approaches as each particular application has a specific set of design requirements and performance metrics.

Video and Image Compression

The general approach used in video compression standards is as follows (Sayood 2017): Each frame is divided up into blocks – often of different sizes. For most of the image frames, a prediction for the block is generated using the previous, and sometimes future, reconstructed frames. The difference is transformed using some variant of the discrete cosine transform (Rao and Yip 1990) and the coefficients encoded using some variant of run-length encoding and an entropy coder. The reconstructed frames to be used for prediction are filtered using a loop filter. The need for hardware assistance in video compression was clear from the time of the initial video compression standards, such as H.261 (ITU-T Recomendation H.261 1993) and MPEG-1 (Mitchell et al. 1997); and hardware implementations of each part of the video coding structure have been proposed. The hardware assistance was initially focused on the discrete cosine transform which is computationally expensive and thus tends to be a bottleneck. A survey of early efforts in this direction can be found in Pirsch et al. (1995). The discrete cosine transform (DCT) is a separable transform (Sayood 2017), which means it can be implemented by taking a one-dimensional transform of the rows followed by a one-dimensional transform of the columns or vice versa. As this is significantly simpler than a direct implementation of the two-dimensional DCT, many of the hardware implementations are for one-dimensional DCT (Martisius et al. 2011; Amer et al. 2005), most based on the efficient Loeffler algorithm (Loeffler et al. 1989). There are also designs for the direct implementation of the two-dimensional DCT (Chang et al. 2000; Gong et al. 2004). The latest standard, variously known as High Efficiency Video Coding, H.265, and MPEG H Part 2, uses a mixture of block sizes and an integer approximation to the DCT. A number of implementations for the transform used in HEVC

have been proposed (Pastuszak 2014; Meher et al. 2014; Park et al. 2012).

Other common components of most video coding algorithms are the loop filter (Cho et al. 2015; Khurana et al. 2006), the motion adaptive prediction, and the entropy coder. The latter has been a particular target for hardware-assisted design as the coder has become more and more complex with each succeeding standard. The most widely deployed entropy coder is the context-adaptive binary arithmetic coder (CABAC). The binary arithmetic coder itself is relatively simple to implement; however, the number of contexts and the approaches for generating the contexts using rate distortion optimization have become increasingly complex and computationally expensive. Hence there is an increase in the number of hardware implementations for the CABAC encoder and decoder (Nunez-Yanez 2006; Penge et al. 2013; Chen and Sze 2015).

In recent years, a number of companies have begun marketing complete hardware solutions targeting HEVC applications. To name a few, Pixelworks has developed and is marketing HEVC Ultra HD System-On-Chip solutions for the consumer electronics industry (www.pixelworks.com), while VITEC is providing end-to-end video streaming solutions that are based on compact and portable HEVC hardware encoders (www.vitec.com). System-On-Chip Technologies offers high-performance HEVC IP Cores that are suitable for integration into midrange Xilinx and Intel FPGAs (www.soc technologies.com).

The video coding standards are targeted to the very important broadcast market. However, there are more specific applications for video compression. One application area that has been particularly active has been CMOS imagers with focal plane compression. Because of the freedom from standards, a number of innovative approaches have been proposed such as Olyaei and Genov (2013), Leon-Salas et al. (2006), Leon-Salas et al. (2007), Chen et al. (2011), Oliveira et al. (2013), Estevo Filho et al. (2013), which use differential encoding and vector quantization, while others (Lin et al. 2008; Cardoso and Gomes

2014; Wang and Leon-Salas 2012) use time-frequency multiresolution approaches.

Wireless Sensor Applications

The constraints on wireless sensor applications have more to do with energy than time. Compression is often essential in wireless sensor network applications as communication costs are substantially higher than processing costs (Anastasi et al. 2009). Despite this advantage, the amount of processing required for compression still puts a high cost in terms of energy use. This is particularly true for visual sensor networks where while it is imperative that the video sequence or images be compressed, the processing costs can be prohibitive. Therefore, it is entirely understandable that this area is ripe for the use of hardware-assisted compression. A survey of many different image compression approaches used in wireless sensor networks can be found in Khalaf Hasan et al. (2014). Mobile sensor networks for environmental monitoring tend to be small and miserly in their use of energy making the use of hardware-assisted compression useful (Sarmiento et al. 2010). Physiological sensors are another obvious target for hardware-assisted compression. Wearable devices such as cardiac monitors (Singh Alvarado et al. 2012; Deepu and Lian 2015; Deepu et al. 2016) which need to function over a long period of time have strict energy constraints which can be best satisfied with hardware-assisted compression. This is also true for sensors which need to function inside the body such as capsules used in wireless endoscopy (Karargyris and Bourbakis 2010). In this, a capsule containing a video sensor (Wahid et al. 2008; Turcza and Dupлага 2013) records images of the gastrointestinal tract as it passes through the patient.

Memory Systems

As programs and datasets have increased in size and complexity, computer memory has

become more and more crowded leading to increased levels of paging. Paging in memory systems can cause a considerable increase in many applications. It has long been recognized that compression in memory systems can help alleviate this problem. Issues that are specific to compression in memory include speed and the heterogeneity of the data to be compressed. To address these problems, IBM developed the IBMLZ1 compression chip (Cheng and Duyanovich 1995) which used the LZ77 (Ziv and Lempel 1977) algorithm and a Huffman coder (Sayood 2017) to provide compression. This algorithm uses the patterns contained in the history of the sequence as a dictionary to compress the current values. This makes the compression scheme locally adaptive and addresses the data heterogeneity issue. In their chip (as in other technologies using dictionary-based coding), the dictionary is stored in content-addressable memory. The first commercial system to use compression to help reduce the memory load was the IBM Memory Expansion Technology (MXT) (Tremaine et al. 2001; Franaszek 2001) which again used the LZ77 algorithm for compression. Latency issues were handled by using a 32 MB cache. Since the appearance of the MXT technology, memory sizes have grown considerably, and the complexities of the LZ77 algorithm can result in high latencies. A number of algorithms have been developed to reduce the latency by using modified forms of the algorithm (Kjelso et al. 1996; Benini et al. 2002; Sardashti and Wood 2014) with the most well-known being the Frequent Pattern Compression (FPC) schemes (Ekman and Stenstrom 2005). More recent approaches, such as Zhao et al. (2015), use a multifaceted approach where the data is compressed using one of several simple, low-latency schemes. An excellent primer on compression in memory can be found in Sardashti et al. (2015). An interesting recent proposal is to do away with the requirement that all contents of memory be accurately preserved (Ranjan et al. 2017). In this approach contents of memory that can be approximated are compressed using lossy compression which

provides substantially more compression than lossless compression.

Conclusion

While most hardware-assisted compression applications fall into the three groups mentioned above, they by no means exhaust the applications of hardware-assisted compression. Consider (Schmitz et al. 2017) which describes a recently developed video processing chip that, not constrained by the standards, uses differential decorrelation to allow processing at rates of 1000 frames per second, or (Amarú et al. 2014) which uses logic operations to provide decorrelation, or a hardware-assisted set partitioning in hierarchical trees (SPIHT) (Said and Pearlman 1996) implementation (Kim et al. 2015). As devices become smaller and more functional, the role of hardware-assisted compression is likely to spread much farther.

H

Cross-References

- ▶ [Data Replication and Encoding](#)
- ▶ [Delta Compression Techniques](#)
- ▶ [Dimension Reduction](#)
- ▶ [Grammar-Based Compression](#)
- ▶ [Graph Compression](#)
- ▶ [In-Memory Transactions](#)

References

- Alvarado AS, Lakshminarayanan C, Principe JC (2012) Time-based compression and classification of heartbeats. *IEEE Trans Biomed Eng* 59(6):1641–1648
- Amarú L, Gaillardon P-E, Burg A, De Micheli G (2014) Data compression via logic synthesis. In: 19th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, pp 628–633
- Amer I, Badawy W, Jullien G (2005) A high-performance hardware implementation of the H.264 simplified 8/spl times/8 transformation and quantization [video coding]. In: IEEE International Conference on acoustics, speech, and signal processing, proceedings (ICASSP'05). , vol 2. IEEE, pp ii–1137
- Anastasi G, Conti M, Francesco MD, Passarella A (2009) Energy conservation in wireless sensor networks: a survey. *Ad Hoc Netw* 7(3):537–568

- Benini L, Bruni D, Macii A, Macii E (2002) Hardware-assisted data compression for energy minimization in systems with embedded processors. In: Proceedings of the conference on design, automation and test in Europe. IEEE Computer Society, pp 449
- Cardoso BB, Gomes JGRC (2014) CMOS imager with focal-plane image compression based on the EZW algorithm. In: 2014 IEEE 5th Latin American symposium on circuits and systems. pp 1–4
- Chang T-S, Kung C-S, Jen C-W (2000) A simple processor core design for DCT/IDCT. *IEEE Trans Circuits Syst Video Technol* 10(3):439–447
- Cheng J-M, Duyanovich LM (1995) Fast and highly reliable IBMLZ1 compression chip and algorithm for storage. In: IEEE [IEE95]. pp 143–154
- Chen YH, Sze V (2015) A deeply pipelined cabac decoder for HEVC supporting level 6.2 high-tier applications. *IEEE Trans Circuits Syst Video Technol* 25(5):856–868
- Chen S, Bermak A, Wang Y (2011) A CMOS image sensor with on-chip image compression based on predictive boundary adaptation and memoryless QTD algorithm. *IEEE Trans Very Large Scale Integr (VLSI) Syst* 19(4):538–547
- Cho S, Kim H, Kim HY, Kim M (2015) Efficient in-loop filtering across tile boundaries for multi-core HEVC hardware decoders with 4K/8K-UHD video applications. *IEEE Trans Multimedia* 17(6):778–791
- Deepu CJ, Lian Y (2015) A joint QRS detection and data compression scheme for wearable sensors. *IEEE Trans Biomed Eng* 62(1):165–175
- Deepu CJ, Zhang XY, Wong DLT, Lian Y (2016) An ECG-on-chip with joint QRS detection & data compression for wearable sensors. In: 2016 IEEE international symposium on circuits and systems (ISCAS). IEEE, pp 2908–2908
- Ekman M, Stenstrom P (2005) A robust main-memory compression scheme. In: ACM SIGARCH computer architecture news, vol 33. IEEE Computer Society, pp 74–85
- Estevo Filho R de M, Gomes JGRC, Petraglia A (2013) Codebook improvements for a CMOS imager with focal-plane vector quantization. In: 2013 IEEE 4th Latin American symposium on circuits and systems (LASCAS), pp 1–4
- Franaszek PA, Heidelberger P, Poff DE, Robinson JT (2001) Algorithms and data structures for compressed-memory machines. *IBM J Res Dev* 45(2):245–258
- Gong D, He Y, Cao Z (2004) New cost-effective VLSI implementation of a 2-D discrete cosine transform and its inverse. *IEEE Trans Circuits Syst Video Technol* 14(4):405–415
- Hasan KK, Ngah UK, Salleh MFM (2014) Efficient hardware-based image compression schemes for wireless sensor networks: a survey. *Wirel Pers Commun* 77(2):1415–1436
- ITU-T Recomendation H.261 (1993) Video codec for audiovisual services at $p \times 64$ kbit/s
- Karargyris A, Bourbakis N (2010) Wireless capsule endoscopy and endoscopic imaging: a survey on various methodologies presented. *IEEE Eng Med Biol Mag* 29(1):72–83
- Khurana G, Kassim AA, Chua TP, Mi MB (2006) A pipelined hardware implementation of in-loop deblocking filter in H.264/AVC. *IEEE Trans Consum Electron* 52(2):536–540
- Kim S, Lee D, Kim H, Truong NX, Kim J-S (2015) An enhanced one-dimensional SPIHT algorithm and its implementation for TV systems. *Displays* 40: 68–77
- Kjelso M, Gooch M, Jones S (1996) Design and performance of a main memory hardware data compressor. In: Beyond 2000: Proceedings of the 22nd EUROMICRO conference hardware and software design strategies, EUROMICRO 96. IEEE, pp 423–430
- Leon-Salas WD, Balkir S, Sayood K, Hoffman MW, Schemm N (2006) A CMOS imager with focal plane compression. In: Proceedings of IEEE international symposium on circuits and systems, ISCAS 2006. IEEE, pp 4–pp
- Leon-Salas WD, Balkir S, Sayood K, Schemm N, Hoffman MW (2007) A CMOS imager with focal plane compression using predictive coding. *IEEE J Solid-State Circuits* 42(11):2555–2572
- Lin Z, Hoffman MW, Leon WD, Schemm N, Balkir S (2008) A CMOS image sensor with focal plane spihit image compression. In: 2008 IEEE international symposium on circuits and systems. pp 2134–2137
- Loeffler C, Ligtenberg A, Moschytz GS (1989) Practical fast 1-D DCT algorithms with 11 multiplications. In: 1989 international conference on acoustics, speech, and signal processing, ICASSP-89. IEEE, pp 988–991
- Martisius I, Birvinskas D, Jusas V, Tamosevicius Z (2011) A 2-D DCT hardware codec based on loeffler algorithm. *Elektronika ir Elektrotechnika* 113(7):47–50
- Mitchell JL, Pennebaker WB, Fogg CE, and LeGall DJ (1997) MPEG video compression standard. Chapman and Hall, London
- Meher PK, Park SY, Mohanty BK, Lim KS, Yeo C (2014) Efficient integer DCT architectures for HEVC. *IEEE Trans Circuits Syst Video Technol* 24(1):168–178
- Nunez-Yanez YL, Chouliaras VA, Alfonso D, Rovati FS (2006) Hardware assisted rate distortion optimization with embedded cabac accelerator for the H.264 advanced video codec. *IEEE Trans Consum Electron* 52(2):590–597
- Oliveira FDVR, Haas HL, Gomes JGRC, Petraglia A (2013) CMOS imager with focal-plane analog image compression combining DPCM and VQ. *IEEE Trans Circuits Syst I: Regular Papers* 60(5):1331–1344
- Olyaei A, Genov R (2007) Focal-plane spatially oversampling cmos image compression sensor. *IEEE Trans Circuits Syst I Regul Pap* 54(1):26–34
- Park J-S, Nam W-J, Han S-M, Lee S-S (2012) 2-D large inverse transform (16×16 , 32×32) for HEVC (high efficiency video coding). *JSTS: J Semicond Technol Sci* 12(2):203–211
- Pastuszak G (2014) Hardware architectures for the H.265/HEVC discrete cosine transform. *IET Image Process* 9(6):468–477

- Pirsch P, Demassieux N, Gehrke W (1995) VLSI architectures for video compression-a survey. Proc IEEE 83(2):220–246
- Peng B, Ding D, Zhu X, Yu L (2013) A hardware cabac encoder for HEVC. In: 2013 IEEE international symposium on circuits and systems (ISCAS2013), pp 1372–1375
- Rao KR, Yip P (1990) Discrete Cosine transform – algorithms, advantages, applications. Academic Press, San Diego
- Ranjan A, Raha A, Raghunathan V, Raghunathan A (2017) Approximate memory compression for energy-efficiency. In: 2017 IEEE/ACM international symposium on low power electronics and design (ISLPED), pp 1–6
- Said A, Pearlman WA (1996) A new fast and efficient coder based on set partitioning in hierarchical trees. IEEE Trans Circuits Syst Video Technol 6:243–250
- Sarmiento D, Pang Z, Sanchez MF, Chen Q, Tenhunen H, Zheng L-R (2010) Mobile wireless sensor system for tracking and environmental supervision. In: 2010 IEEE international symposium on industrial electronics (ISIE). IEEE, pp 470–477
- Sardashti S, Wood DA (2014) Decoupled compressed cache: exploiting spatial locality for energy optimization. IEEE Micro 34(3):91–99
- Sardashti S, Arellakis A, Stenström P, Wood DA (2015) A primer on compression in the memory hierarchy. Synthesis Lect Comput Archit 10(5):1–86
- Sayood K (2017) Introduction to data compression, 5th edn. Morgan Kauffman-Elsevier, San Francisco
- Schmitz JA, Gharzai MK, Balkir S, Hoffman MW, White DJ, Schemm N (2017) A 1000 frames/s vision chip using scalable pixel-neighborhood-level parallel processing. IEEE J Solid-State Circuits 52(2):556–568
- Turcza P, Dupлага M (2013) Hardware-efficient low-power image processing system for wireless capsule endoscopy. IEEE J Biomed Health Inf 17(6): 1046–1056
- Tremaine RB, Franaszek PA, Robinson JT, Schulz CO, Smith TB, Wazlowski ME, Bland PM (2001) IBM memory expansion technology (MXT). IBM J Res Dev 45(2):271–285
- Wahid K, Ko S-B, Teng D (2008) Efficient hardware implementation of an image compressor for wireless capsule endoscopy applications. In: IEEE international joint conference on neural networks, IJCNN 2008. IEEE World congress on computational intelligence. IEEE, pp 2761–2765
- Wang HT, WD-Salas L (2012) A multiresolution algorithm for focal-plane compression. In: 2012 IEEE international symposium on circuits and systems. pp 926–929
- Ziv J, Lempel A (1977) A universal algorithm for data compression. IEEE Trans Inf Theory IT-23(3): 337–343
- Zhao J, Li S, Chang J, Byrne JL, Ramirez LL, Lim K, Xie Y, Faraboschi P (2015) Buri: scaling big-memory computing with hardware-based memory expansion. ACM Trans Archit Code Optim (TACO) 12(3):31

Hardware-Assisted Transaction Processing

Many-core

Pinar Tözün

IT University of Copenhagen, Copenhagen, Denmark

Synonyms

OLTP on modern hardware; Scaling up OLTP on multicores and many cores; Transaction processing on modern and emerging hardware

H

Definitions

Online transaction processing (OLTP) is one of the most important and demanding database applications. A **transaction** is a unit of work that satisfies the **ACID** properties (Gray and Reuter 1992). Atomicity ensures that when a transaction ends, either all or none of its effects are visible to the other transactions. Consistency guarantees that the effect of a transaction transforms the database from one consistent state to another. Isolation property gives the illusion that transactions do not interfere with each other's effects to the database even if they run concurrently. Durability assures that the effects of complete transactions must be persistent in the database. The maintenance of these properties in the face of many concurrent client requests is a big challenge and complicates the design of transaction processing systems. Multiple components of a transaction processing system are involved in satisfying each of the ACID properties, and a component is usually involved in providing several properties. These tightly coupled components lead to various forms of underutilization on multicore processors.

Multicores refer to processors with two or more independent processing units (i.e., cores). **Many cores** are a specialized type of multicores, where the parallelism is more extensive (100s

or 1000s of cores in a processor). Today's commodity server hardware where majority of data management applications, including OLTP, run on typically has multiple multicore processors. Such hardware is called **multisocket multicore** hardware.

Overview

Transaction processing has been at the forefront of many influential advancements in the data management ecosystem (Diaconu et al. 2013; Exadata 2015; Kongetira et al. 2005; Stonebraker et al. 2007). In the recent years, many of these advancements have been triggered by the developments in the computer architecture community: multicore parallelism, large main memories, hardware transactional memory, nonvolatile memory, remote direct memory access, etc. These developments led to rethinking of the overall system design for transaction processing systems. In turn, several specialized transaction processing systems have evolved adopting a variety of system designs each with their unique pros/cons. The focus of this entry is to survey the techniques proposed in recent years to scale up transaction processing on modern multicore hardware and assess how suitable these techniques are in the context of emerging many-core hardware and trends in the computer architecture community in general.

Up until around 2005, software systems benefited from the advancements in the hardware industry easily. Thanks to Moore's law (Moore 1965), computer architects were able to boost the performance of processors by adding more and more complexity within a core (e.g., aggressive pipelining, super-scalar execution, out-of-order execution, and branch prediction). Around 2005, however, power draw and heat dissipation have prevented processor vendors from leaning on more complex micro-architectural techniques for higher performance. Instead, they have started adding more cores on a single processor, creating multicores, to enable exponentially increasing parallelism (Olukotun et al. 1996). As a result,

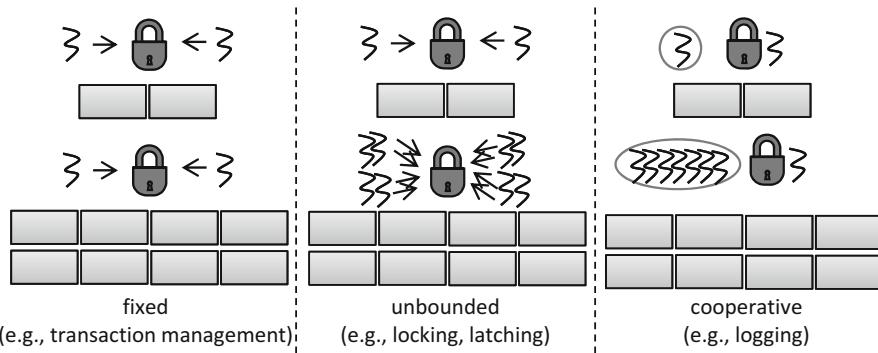
software developers did not have the luxury to be blind to the advancements in hardware. The processors, and in turn software systems, did not automatically get faster with each new generation of hardware. Traditional software systems, including transaction processing, had to go through fundamental design changes to be able to exploit the kind of parallelism offered by having multiple cores in a processor.

The traditional transaction processing systems were not designed with abundant horizontal parallelism in mind. Even though they can handle several concurrent requests very efficiently while satisfying the *ACID* properties on a few single-core processors, their performance suffers on multicore architectures due to the tightly coupled and centralized internal components and unpredictable accesses to the shared data (Johnson et al. 2014). Such components and access patterns lead to various scalability bottlenecks on multicores and hinder the parallel execution of even non-conflicting requests.

In the rest of this entry, we first categorize the communication patterns and, in turn, types of critical sections, in transaction processing to better understand the underlying scalability problems. Then, we review the recent works that tackle the scalability of transaction processing systems proposing novel system designs that are based on data partitioning and that avoid data partitioning. Finally, we discuss the issues that await modern transaction processing systems on emerging hardware and identify the significant research challenges in this context and conclude the entry.

Communication Patterns in Transaction Processing

To analyze the communication patterns of a transaction processing system, we are going to look at the types of critical sections it executes. Transaction processing systems employ several types of communication and synchronization. *Locking* operates at the logical (application) level to enforce isolation and atomicity among transactions.



Hardware-Assisted Transaction Processing, Fig. 1 Communication patterns in transaction processing based on the type of contention they create as the parallelism increases (Johnson et al. 2014)

Latching operates at the physical (database page) level to enforce the consistency of the physical data as concurrent transactions read/write this data. Finally, at the lowest levels, *critical sections* ensure the consistency of the system's internal state. Locks and latches form a crucial part of that internal state and are themselves protected by critical sections underneath. Therefore, analyzing the behavior of critical sections captures nearly all forms of communication in a transaction processing system.

Critical sections, in turn, fall into three categories depending on the nature of the contention they tend to cause in the system as Fig. 1 illustrates. *Fixed* critical sections are the ones where a fixed number of threads use to protect their communication. Therefore, the contention is limited, and it cannot increase as the underlying parallelism increases. Producer-consumer-like communication, mainly found inside the transaction manager component of a transaction processing system, is an example for these types of critical sections. At the other extreme, *unbounded* critical sections are the ones where all concurrent worker threads in the system might contend for. Therefore, as hardware parallelism increases, the degree of contention also has the potential to increase. Such critical sections inevitably grow into a bottleneck. Making them shorter or less frequent provides a little slack but does not fundamentally improve scalability. Locking and latching are good examples for this category of critical sections. Finally, *cooperative* critical sections are

the ones where concurrent worker threads can aggregate their requests and execute them all at once later. They are not as lightweight as the fixed critical sections. However, they allow good resistance against contention, since adding more threads to the system gives more opportunity for threads to combine work rather than competing directly for the critical section. Batching log requests is an example of cooperative communication in a transaction processing system.

The real key to scalability lies in converting all unbounded communication to either the fixed or cooperative type, thus removing the potential for bottlenecks to arise. The following sections analyze the various proposals for building more scalable transaction processing systems in the context of this categorization of communication patterns and critical sections. Note that this categorization is not unique to data management systems and can be used to reason about the scalability of other types of software systems as well.

Key Research Findings

Proposals that depart from traditional transaction processing in order to scale up on multicores can be grouped into two broad categories. The first category applies some sort of data partitioning, physical or logical, to bound the concurrent accesses on data. The second category prefers not depending on data partitioning and in-

stead relies on lighter-weight concurrency control mechanisms, lock-free algorithms, and multiversioning. The former has the potential to eliminate the most problematic critical sections in a system but requires the data to be almost perfectly partitionable. The latter doesn't require the data to be perfectly partitionable but doesn't eliminate the critical sections either; it instead reduces/minimizes their impact. More importantly, both approaches have challenges when it comes to scaling up on multisocket multicore (Porobic et al. 2012) or many-core hardware (Yu et al. 2014).

Scaling Up with Partitioning

There are various ways to partition the data from extreme physical partitioning to lightweight logical partitioning to more hybrid partitioning that combines physical and logical partitioning. This section briefly surveys these partitioning mechanisms going over some of the systems that adopt them.

Physical Partitioning

Shared-nothing systems physically partition the data and deploy multiple database instances. In this setup, one can tune which part of the data belongs to which instance and the number of threads assigned to each instance. Therefore, physically partitioned systems can regulate the data access patterns and contention on the data. Systems like VoltDB (<http://www.voltdb.com>) (commercial version of H-Store (Stonebraker et al. 2007)) apply the idea of physical partitioning to the extreme and deploy single-threaded database instances. When a workload is perfectly partitionable, i.e., all the transactions are handled by a single database instance and data accesses are balanced across the instances, this design eliminates all unbounded communication within the database engine and has the potential to achieve perfect scalability.

When a workload is not amenable to partitioning, though, physical partitioning suffers from multi-partition/distributed transactions (Curino et al. 2010; Helland 2007; Pavlo et al. 2011; Porobic et al. 2012) or load imbalance (Pavlo et al. 2012). Dynamic repartitioning to minimize

distributed transactions or balance load is highly costly due to the amount of data to be moved from one partition to another and reorganization of the index structures (Serafini et al. 2014; Tözün et al. 2013). In addition, traditional physiological logging (Mohan et al. 1992) does not favor this design since maintaining a partitioned log has high overhead, especially for non-partitionable workloads (Johnson et al. 2012). Therefore, physically partitioned systems either use a shared log (Lomet et al. 1992) or eliminate it completely (Stonebraker et al. 2007) and provide durability through replication or adopt more lightweight forms of logging (Malviya et al. 2014).

Logical or Physiological Partitioning

Instead of physically partitioning the data and adopting a shared-nothing design, one can also just logically partition the data in a shared-everything setting to regulate the data access patterns and the contention over the shared data. More specifically, logical partitioning decentralizes the lock manager and enforces each partition to maintain its own lock manager. Each logical partition can be owned by either a single worker thread (Pandis et al. 2010) or a few worker threads (Lahiri et al. 2001). This design eliminates the unbounded communication due to locking. In addition, dynamic repartitioning is cheap under logical partitioning since no data movement is necessary (Tözün et al. 2013).

On the other hand, logical partitioning is not able to regulate the accesses to shared database pages. Therefore, it cannot eliminate the unbounded communication due to latching. Physiological partitioning (Pandis et al. 2011; Schall and Härdler 2015) enhances logical partitioning and partitions physical data page accesses as well. This type of partitioning can eliminate the unbounded communication due to both locking and latching while still reaping the benefits of a shared-everything setting.

However, both logical and physical partitioning introduce multi-partition transactions in a shared-everything system. Such transactions can be handled by some extra fixed communication to coordinate the actions of a transaction that belong to different partitions.

While such fixed communication is usually lightweight and is not a burden for scalability as section “Communication Patterns in Transaction Processing” describes, it has to be handled within a processor socket as much as possible. Otherwise, such fixed communication can easily become a bottleneck as well due to nonuniform memory accesses as Porobic et al. (2014) shows and as section “Future Directions of Research: Toward Many Cores” elaborates.

Scaling Up Without Partitioning

While partitioning-based techniques have been quite popular, the recent years have seen the rise of systems work that relies on lock-free techniques while maintaining the consistency of their internal data structures and optimistic and multiversion concurrency control to ensure isolation and atomicity of concurrent transactions. There are various commercial transaction processing systems that follow this approach: HyPer (Kemper et al. 2013), Hekaton (Diaconu et al. 2013), MemSQL (<http://www.memsql.com/>), SAP HANA (Lee et al. 2013), etc. In addition to typical OLTP workloads, these systems also benefit workloads that have short-running read/write transactions concurrently running with the long-running read-only transactions or queries or workloads that need hybrid transactional/analytical processing (HTAP), since they do not block transactions with pessimistic locking techniques.

However, these systems do not eliminate the unbounded communication at all. They rather minimize the time spent for such communication through optimistic concurrency control, decentralized record locks, frequent use of atomic operations, read-copy-update mechanisms, or hardware transactional memory (Larson et al. 2011; Leis et al. 2014; Levandoski et al. 2013; Sewall et al. 2011; Tu et al. 2013). Even though these mechanisms perform and scale well on architectures with a few (one to four) processor sockets without depending on data partitioning, scaling them up on larger multisocket multicore hardware or emerging many-core processors is not straightforward (David et al. 2013; Porobic et al. 2014; Yu et al. 2014). In addition, similarly to the statically partitioned designs, they suffer under

update-heavy workloads with hotspots (Narula et al. 2014).

Future Directions of Research: Toward Many Cores

Commodity servers that data-intensive applications typically run on have been offering us more and more opportunities for parallelism over the years, thanks to Moore’s law, even though its form and level have been changing over time. As mentioned in section “Overview”, this parallelism has been mainly implicit (more complexity within a core) shortly before the last decade. Multicores, on the other hand, offer explicit parallelism where tasks can run concurrently. *A software system has to exploit both types of parallelism if one wants to utilize modern hardware fully.*

Nowadays we tend to have multiple multicore processors in server hardware, typically two to four sockets on commodity servers. Such hardware does not offer uniform core-to-core communication anymore. Cores within a processor socket tend to communicate at least an order of magnitude faster compared to the cores on different sockets (David et al. 2013) creating nonuniform communication within a server. In the future, we expect even more cores in a processor and even more processors in commodity servers. Core-to-core access latency is going to be nonuniform even within a processor socket. *Therefore, system designs targeting only parallelism and not nonuniformity of communication across cores are not ideal.*

Unfortunately, the traditional multicore hardware design also faces challenges. Even though Moore’s law still holds today, Dennard scaling (Dennard et al. 1974), which enables keeping the power density of the transistors constant, does not. The supply voltage required to power all the transistors up does not decrease at a proportional rate (Esmailzadeh et al. 2011). Putting more cores in a processor rather than putting more complexity within a core helped mitigating this problem up until now, but is not going to help us solve this problem any longer. Even if we end up

incorporating more cores on a single processor, we are not going to be able to use them all at the same time. This trend is called *dark silicon* and fundamentally alters the focus of hardware designs (Hardavellas et al. 2011). *In this new era, the focus has to shift toward optimizing energy per instruction.*

This focus on energy and the ever-growing scale of data-intensive applications has increased the interest in building hardware components that are specialized for data-intensive processing tasks. Most of the efforts in this direction so far has targeted traditional analytics workloads (Balkesen et al. 2018; Wu et al. 2014) or cloud search, machine learning, and AI (Putnam et al. 2014; Jouppi et al. 2017) more recently. There have been efforts for building specialized hardware components for transaction processing as well (Johnson and Pandis 2013). *Even though it might still not be feasible to design a whole chip specifically for a data-intensive application, building hardware components that accelerate frequent operations from an application might be, especially if the accelerator can benefit similar operations from other applications as well. The challenge is to find such operations that would justify the cost to build specialized hardware.*

Alternatively, many hardware vendors have started to ship co-processors. Low power processing units shipped together with power hungry ones or regular CPUs are shipped together with FPGAs or GPUs on the side (<https://www.arm.com/products/processors/biglittleprocessing.php>) (<https://www.altera.com/solutions/acceleration-hub/overview.html>) (<https://www.ibm.com/support/knowledgecenter/en/POWER9/p9hdx/POWER9welcome.htm>). *Finding ways to exploit such heterogeneous platforms is going to be another big challenge for data-intensive systems.*

Conclusions

Despite various advancements both from academia and industry, tackling ever-increasing parallelism and heterogeneity on modern and emerging hardware is still a hot challenge for

transaction processing systems. None of the major proposals for concurrency control are good enough for many-core hardware (~ 1000 cores) (Yu et al. 2014) even when one assumes uniform cores. As long as there is unbounded communication in a system, even the most scalable systems on the current generation of hardware will fail to scale up in the future generations. In addition, scheduling tasks on today's multisocket hardware or on hardware with mild specialization is not straightforward. Focusing on parallelism as the only dimension for scaling up transaction processing systems is insufficient. We have to take nonuniformity and heterogeneity of the underlying hardware into account.

More and more emerging hardware is going to be nonuniform and heterogeneous in terms of the provided functionality. Therefore, it also becomes essential for the emerging transaction processing systems to decide on the optimal design options based on the processor types they are running on. If the system is running on a hardware that has a combination of different processing units (aggressive, low-power, special purpose, etc.), it should be able to automatically decide on which types of requests should run on which types of processors. If the system is running on a hardware that has support for hardware transactional memory, it should be able to know the types of critical sections that would benefit from switching the synchronization primitive to transactions and the ones that should keep its existing synchronization method. If there are tasks that would benefit from running on GPUs or could be accelerated via FPGAs, the system should know when to off-load these tasks to these units. Nevertheless, this requires a thorough understanding of the specific requirements of a system that can exploit the specific features of various hardware types. In addition, it requires interdisciplinary collaborations, especially involving people from data management, computer architecture, and compiler communities. Furthermore, to come up with more economically viable solutions in this area, we need to reach out to cloud providers so that they start building

software and hardware infrastructures with heterogeneity in mind.

Cross-References

- ▶ Emerging Hardware Technologies
- ▶ Hardware-Assisted Transaction Processing: NVM
- ▶ In-Memory Transactions

References

- Balkesen C, Kunal N, Giannikis G, Fender P, Sundara S, Schmidt F, Wen J, Agrawal S, Raghavan A, Varadarajan V, Viswanathan A, Chandrasekaran B, Idicula S, Agarwal N, Sedlar E (2018) A many-core architecture for in-memory data processing. In: SIGMOD
- Curino C, Jones E, Zhang Y, Madden S (2010) Schism: a workload-driven approach to database replication and partitioning. *PVLDB* 3:48–57
- David T, Guerraoui R, Trigonakis V (2013) Everything you always wanted to know about synchronization but were afraid to ask. In: SOSP, pp 33–48
- Dennard RH, Gaensslen FH, Yu HN, Rideout VL, Bassous E, Leblanc AR (1974) Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE J Solid-State Circuits* 9:256–268
- Diaconu C, Freedman C, Ismert E, Larson PA, Mittal P, Stonecipher R, Verma N, Zwilling M (2013) Hekaton: SQL server's Memory-optimized OLTP engine. In: SIGMOD, pp 1243–1254
- Esmaeilzadeh H, Blehm E, St Amant R, Sankaralingam K, Burger D (2011) Dark silicon and the end of multicore scaling. In: ISCA, pp 365–376
- Exadata (2015) Oracle corp.: exadata database machine. <http://www.oracle.com/technetwork/database/exadata-overview/index.html>
- Gray J, Reuter A (1992) Transaction processing: concepts and techniques. Morgan Kaufmann Publishers Inc., San Francisco
- Hardavellas N, Ferdman M, Falsafi B, Ailamaki A (2011) Toward dark silicon in servers. *IEEE Micro* 31(4):6–15
- Helland P (2007) Life beyond distributed transactions: an apostate's opinion. In: CIDR, pp 132–141
- Johnson R, Pandis I (2013) The bionic DBMS is coming, but what will it look like? In: CIDR
- Johnson R, Pandis I, Stoica R, Athanassoulis M, Ailamaki A (2012) Scalability of write-ahead logging on multi-core and multisocket hardware. *Vldb J* 21:239–263
- Johnson R, Pandis I, Ailamaki A (2014) Eliminating unscalable communication in transaction processing. *Vldb J* 23(1):1–23
- Jouppi NP, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A, Boyle R, Cantin P, Chao C, Clark C, Coriell J, Daley M, Dau M, Dean J, Gelb B, Ghaemmaghami TV, Gottipati R, Gulland W, Hagmann R, Ho CR, Hogberg D, Hu J, Hundt R, Hurt D, Ibarz J, Jaffey A, Jaworski A, Kaplan A, Khaitan H, Killebrew D, Koch A, Kumar N, Lacy S, Laudon J, Law J, Le D, Leary C, Liu Z, Lucke K, Lundin A, MacKean G, Maggiore A, Mahony M, Miller K, Nagarajan R, Narayanaswami R, Ni R, Nix K, Norrie T, Omernick M, Penukonda N, Phelps A, Ross J, Ross M, Salek A, Samadiani E, Severn C, Sizikov G, Snelham M, Souter J, Steinberg D, Swing A, Tan M, Thorson G, Tian B, Toma H, Tuttle E, Vasudevan V, Walter R, Wang W, Wilcox E, Yoon DH (2017) In-datacenter performance analysis of a tensor processing unit. In: ISCA, pp 1–12
- Kemper A, Neumann T, Finis J, Funke F, Leis V, Mühe H, Mühlbauer T, Rödiger W (2013) Transaction processing in the hybrid OLTP&OLAP main-memory database system HyPer. *IEEE DEBull* 36(2):41–47
- Kongetira P, Aingaran K, Olukotun K (2005) Niagara: a 32-way multithreaded sparc processor. *IEEE Micro* 25(2):21–29
- Lahiri T, Srihari V, Chan W, MacNaughton N, Chandrasekaran S (2001) Cache fusion: extending shared-disk clusters with shared caches. In: VLDB, pp 683–686
- Larson PA, Blanas S, Diaconu C, Freedman C, Patel JM, Zwilling M (2011) High-performance concurrency control mechanisms for main-memory databases. *PVLDB* 5(4):298–309
- Lee J, Kwon YS, Farber F, Muehle M, Lee C, Bensberg C, Lee JY, Lee A, Lehner W (2013) SAP HANA distributed in-memory database system: transaction, session, and metadata management. In: ICDE, pp 1165–1173
- Leis V, Kemper A, Neumann T (2014) Exploiting hardware transactional memory in main-memory databases. In: ICDE, pp 580–591
- Levandoski J, Lomet D, Sengupta S (2013) The Bw-tree: a B-tree for new hardware platforms. In: ICDE, pp 302–313
- Lomet D, Anderson R, Rengarajan TK, Spiro P (1992) How the Rdb/VMS data sharing system became fast. Technical Report CRL-92-4, DEC
- Malviya N, Weisberg A, Madden S, Stonebraker M (2014) Rethinking main memory OLTP recovery. In: ICDE, pp 604–615
- Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P (1992) ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM TODS* 17(1): 94–162
- Moore G (1965) Cramming more components onto integrated circuits. *Electronics* 38(6):82–85
- Narula N, Cutler C, Kohler E, Morris R (2014) Phase reconciliation for contended in-memory transactions. In: OSDI, pp 511–524

- Olukotun K, Nayfeh BA, Hammond L, Wilson K, Chang K (1996) The case for a single-chip multiprocessor. In: ASPLOS, pp 2–11
- Pandis I, Johnson R, Hardavellas N, Ailamaki A (2010) Data-oriented transaction execution. PVLDB 3(1):928–939
- Pandis I, Tözün P, Johnson R, Ailamaki A (2011) PLP: page latch-free shared-everything OLTP. PVLDB 4(10):610–621
- Pavlo A, Jones EPC, Zdonik S (2011) On predictive modeling for optimizing transaction execution in parallel OLTP systems. PVLDB 5(2):85–96
- Pavlo A, Curino C, Zdonik S (2012) Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In: SIGMOD, pp 61–72
- Porobic D, Pandis I, Branco M, Tözün P, Ailamaki A (2012) OLTP on hardware Islands. PVLDB 5(11):1447–1458
- Porobic D, Liarou E, Tözün P, Ailamaki A (2014) ATraPos: adaptive transaction processing on hardware Islands. In: ICDE, pp 688–699
- Putnam A, Caulfield A, Chung E, Chiou D, Constantinides K, Demme J, Esmaeilzadeh H, Fowers J, Gopal GP, Gray J, Haselman M, Hauck S, Heil S, Hormati A, Kim JY, Lanka S, Larus J, Peterson E, Pope S, Smith A, Thong J, Xiao PY, Burger D (2014) A reconfigurable fabric for accelerating large-scale datacenter services. In: ISCA, pp 13–24
- Schall D, Härdter T (2015) Dynamic physiological partitioning on a shared-nothing database cluster. In: ICDE, pp 1095–1106
- Serafini M, Mansour E, Aboulnaga A, Salem K, Taha R, Minhas UF (2014) Accordion: elastic scalability for database systems supporting distributed transactions. PVLDB 7(12):1035–1046
- Sewall J, Chhugani J, Kim C, Satish N, Dubey P (2011) PALM: parallel architecture-friendly latch-free modifications to B+Trees on many-core processors. PVLDB 4(11):795–806
- Stonebraker M, Madden S, Abadi DJ, Harizopoulos S, Hachem N, Helland P (2007) The end of an architectural era: (it's time for a complete rewrite). In: VLDB, pp 1150–1160
- Tözün P, Pandis I, Johnson R, Ailamaki A (2013) Scalable and dynamically balanced shared-everything OLTP with physiological partitioning. VLDB J 22(2): 151–175
- Tu S, Zheng W, Kohler E, Liskov B, Madden S (2013) Speedy transactions in multicore in-memory databases. In: SOSP, pp 18–32
- Wu L, Lottarini A, Paine TK, Kim MA, Ross KA (2014) Q100: the architecture and design of a database processing unit. In: ASPLOS, pp 255–268
- Yu X, Bezerra G, Pavlo A, Devadas S, Stonebraker M (2014) Staring into the abyss: an evaluation of concurrency control with one thousand cores. PVLDB 8(3):209–220

Hardware-Assisted Transaction Processing: NVM

Ilia Petrov¹, Andreas Koch², Tobias Vinçon¹, Sergey Hardock³, and Christian Rieger¹

¹Data Management Lab, Reutlingen University, Reutlingen, Germany

²Embedded Systems and Applications Group, TU Darmstadt, Darmstadt, Germany

³Databases and Distributed Systems Group, TU Darmstadt, Darmstadt, Germany

Synonyms

Transaction processing on persistent memory;
Transaction processing on storage-class memory

Definitions

A transaction is a demarcated sequence of application operations, for which the following properties are guaranteed by the underlying transaction processing system (TPS): atomicity, consistency, isolation, and durability (ACID). Transactions are therefore a general abstraction, provided by TPS that simplifies application development by relieving transactional applications from the burden of concurrency and failure handling. Apart from the ACID properties, a TPS must guarantee high and robust performance (high transactional throughput and low response times), high reliability (no data loss, ability to recover last consistent state, fault tolerance), and high availability (infrequent outages, short recovery times).

The architectures and workhorse algorithms of a high-performance TPS are built around the properties of the underlying hardware. The introduction of nonvolatile memories (NVM) as novel storage technology opens an entire new problem space, with the need to revise aspects such as the virtual memory hierarchy, storage management and data placement, access paths, and indexing. NVM are also referred to as storage-class memory (SCM).

Overview

Over the last decades, advances in the semiconductor industry regarding novel storage technologies and the resulting disruptive applications have been remarkable. Even though databases or file systems still do not fully exploit NVM properties, nonvolatile memory devices are gradually entering the mass market. There are several promising NVM technologies, such as resistive memory (e.g., HP's memristor), spin-transfer torque memory (STT-RAM), Intel's 3D XPoint, and phase-change memories (PCM). Their characteristics differ considerably from existing storage solutions. Similar to Flash, NVMs wear out and exhibit asymmetric I/O latencies; similar to DRAM, they are byte-addressable, but they are much denser and only consume energy during I/O activity (proportional to the data size to be written), and their I/O latencies are 2–50× slower than DRAM yet about 100× faster than Flash (see Table 1).

Key Research Findings

The advent of modern storage technologies since 2006 resulted in novel approaches and algorithms on virtually all layers of modern DBMS: storage and transaction management, access paths and indexing, and query processing. Since transactions commonly traverse these layers, current research shows the significant impact of using these advances in the DBMS processing stack.

Architectural Considerations

The architectural positioning of NVM alongside DRAM yields complex virtual memory hierarchies (Fig. 1). They comprise block and byte-addressable storage and a combination of wear-prone and wear-immune technologies. Traditional DBMS are designed to compensate for the so-called access gap. Yet, in complex memory hierarchies, this gap turns into an operation-based latency *continuum*, due to read/write asymmetry and different technological combinations. Given all of the above, the research space is extended by fields such as query optimization for asymmetric memories and byte addressability, data placement, and automated advisors, as well as energy efficiency.

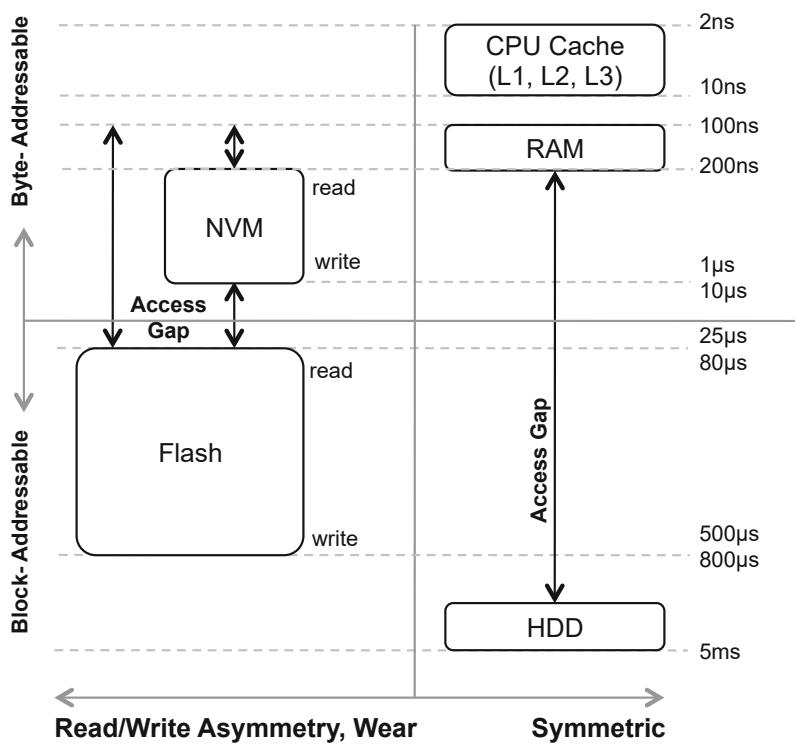
The announced form factors of leading semiconductor manufacturers comprise almost the complete spectrum of currently available peripheral devices and embedded systems. NVM can act as caches in front of NAND Flash (Intel Optane SSD DC P4800X Series), as stand-alone PCIe storage cards (Intel Optane SSD 900P), or even as memory DIMM replacements, directly attached to various processing units (CPU, GPU, FPGA, or ASIC). Besides NVDIMM-F, NVDIMM-N, and various other standards toward novel storage technologies, JEDEC currently defines a new NVDIMM-P standard (JEDEC 2017) for compatibility with traditional DDR4 DIMM slots, which is expected to be released soon. Moreover, its application in computer system either as peripheral, DIMMs, or embedded is widespread and comprises

Hardware-Assisted Transaction Processing: NVM, Table 1

Comparison of storage technologies. (Source ITRS 2011)

	DRAM	PCM	STT-RAM	Memristor	NAND Flash	HDD
Write energy [pJ/bit]	0.004	6	2.5	4	0.00002	10×10^9
Endurance	$>10^{16}$	$>10^8$	$>10^{15}$	$>10^{12}$	$>10^4$	$>10^4$
Page size	64 B	64 B	64 B	64 B	4–16 KB	512 B
Page read latency	10 ns	50 ns	35 ns	<10 ns	~25 us	~5 ms
Page write latency	10 ns	500 ns	100 ns	20–30 ns	~200 us	~5 ms
Erase latency	N/A	N/A	N/A	N/A	~2 ms	N/A
Cell area [μm^2]	6	4–16	20	4	1–4	N/A

Hardware-Assisted Transaction Processing: NVM, Fig. 1 Complex memory hierarchy



small IoT devices, smartphones, consumer devices, but also large data centers. However, the definitive position is still unclear. NVM can function as a RAM replacement but can also be exposed as conventional block device. The former calls for persistent memory controllers with a consistent energy-backed write queue (WQ) as well as extended CPU instruction set (e.g., CLWB, CLFLUSH, etc.). The latter promotes faster market introduction at the cost of underutilizing the technological potential and employing suboptimal backward compatibility.

Furthermore, a variety of different and novel interface definitions lately emerged. Depending on the targeted utilization and application, these approaches focus on various aspects of NVM characteristics. A broad range of lightweight file systems (PMFS (Dulloor et al. 2014), NOVA (Xu and Swanson 2016), etc.) including kernel drivers and user mode libraries were proposed in the last half decade. But also a potpourri of novel access mechanisms for nonvolatile storage (PAllocator Oukid et al. (2017), pmemlib Intel (2017), etc.), adapting and extending conventional concepts of

memory management, are topics of the current research. Arulraj et al. (2015) propose a NVM allocator extension providing a naming mechanism that ensures validity of memory address pointers and pointers even after restart, hence, providing *nonvolatile pointers*. The allocators may also offer *atomicity* (Schwalb et al. 2015) by avoiding dangling pointers since NVM leaks upon a restart.

Moreover, CPU instruction set architectures (ISA) have to be extended by new primitives like CLWB, CLFLUSH, or SFENCE (Oukid and Lehner 2017; Arulraj and Pavlo 2017) to handle NVM durability in combination with the WQ. To avoid consistency issues, modified cache lines need to be written *in order*. ISA extensions for this purpose are necessarily needed, since data modifications caused by a transaction may still rely in the CPU caches and may not directly flushed to the NVM, causing durability and consistency issues in case of a power loss. Supporting some type of soft or atomic writes natively on NVM is an intensive research area (Schwalb et al. 2015; Oukid et al. 2017; Intel 2017).

Storage Manager

The interaction with NVM in the context of transaction is in many cases realized as any kind of storage management. The associated research comprises data placement via file systems or allocators, interface extensions, as well as cost models and advisers for these concepts.

Data Placement in Complex Memory Hierarchies

With NVM as novel, persistent but also fast storage layer, data placement emerges as a pressing research question: *Where* should the data of a database object be placed (completely or partially), so that data access patterns match the characteristics of the respective storage technology, to achieve optimal performance under the current workload? DRAM, for instance, can be utilized for rapidly changing data or as a fast NVM cache to achieve write and energy optimizations as well as compression. NVM storage is predestined for large hot persistent data sets due to its high density and low latencies. Flash has higher access latencies but increasing large storage capacity (with the advent of 3D NAND) and can serve as cold or read-intensive storage. HDD can serve as cheap cold storage devices for large sequentially accessed data sets.

The extent to which NVM can function as a DRAM replacement is evaluated in Huang et al. (2014), concluding that a replacement becomes quickly inefficient as soon as transaction-heavy benchmarks like TPC-C, TATP, or TPC-B are executed. To overcome this, novel hybrid NVM-DRAM storage engines like FOEDUS, (Kimura 2015) or SOFORT, (Oukid et al. 2014) and respective data structures, algorithms, and hybrid page layouts are subjects under research.

Hardock et al. (2017) exploit low-level techniques on Flash storage to handle appends to already programmed physical pages. Extending the traditional NSM page layout with a delta record area and proposing a scheme to control the write behavior and space allocation resulted in not only significantly less reads, writes, and erases but also in a higher transactional throughput.

Still, data placement in complex memory hierarchies does not influence exclusively the trans-

action throughput. In addition, a wide array of performance metrics like recovery time, as studied in DeBrabant et al. (2014a). Viglas (2015) and Arulraj et al. (2015) evaluate and identify further open problems in various traditional DBMS, e.g., validating in-place, copy-on-write, and log structured updates. In this context, Ma et al. (2016) explore issues regarding swapping in in-memory databases to overcome data size limitations, especially with NVM. This includes policies for when to evict tuples and how to bring them back. Reconsiderations of the OLTP durability management for NVM is presented in Pelley et al. (2013a). NVM as disk replacement for near-instantaneous recovery and the removal of software-managed DRAM buffering is presented. Mihnea et al. (2017) have already shown an early adoption of NVM within SAP HANA.

Access Paths and Interface Extensions

The use of the file abstraction and lightweight file systems (FS) for NVM is another intensively investigated research area. First approaches to bypass the slow, block-based interface of conventional FS are introduced in Condit et al. (2009). Their new technique *short-circuit shadow paging* supports atomic, fine-grained updates while enabling strong reliability guarantees. Dulloor et al. (2014) set their focus with PMFS likewise on the byte addressability and the avoidance of block-oriented overheads. Thereby, they exploit processor's paging and memory-order features for optimization and use fine-grained logging to ensure consistency. By changing the architecture for user-mode programs to access files without kernel interaction, Volos et al. (2014) present a novel and flexible FS with fewer consistency guarantees but higher performance. Xu and Swanson (2016) adapt commonly used log-structuring techniques with their NOVA FS and provide strong consistency guarantees on hybrid memory systems. Having lightweight characteristics in common, all these FSs are claimed to facilitate storage management on modern architectures.

An alternative approach is In-Page Logging for Flash and NVM, which allocates a small log sector of a page on the same Flash block to

accumulate updates to the respective page. In addition, partial differential logging of Kim et al. (2010) explores the idea of reducing write amplification. Many of these principles have been extended and applied on storage management for NVM (Lee et al. 2010a,b; Gao et al. 2015; Kim et al. 2011). To further minimize the write volume and perform NVM-friendly writes, storage managers use techniques such as epochs, change ordering, and group commit (Pelley et al. 2013b).

Volos et al. (2011) address the interface to modern storage technologies with Mnemosyne and target the creation and management of NVM as well as consistency handling in the presence of failures. Likewise, Intel (2017) defines a set of libraries designed to provide some useful API server applications, e.g., a transactional object store.

REWIND, Chatzistergiou et al. (2015a), proposes a user-mode library to manage transactional updates *directly* in the application to minimize the overhead of the conventional access stack. PAllocator is proposed in Oukid et al. (2017) as a fail-safe persistent NVM allocator with a special focus on high concurrency, capacity, and scalability. Ogleari et al. (2016) presents an approach that goes one step further in that it handles a sort of undo/redo log directly in hardware.

Transaction Management and Logging

Due to their I/O characteristics, both Flash and NVM are claimed to speed up logging. Numerous methods have been developed for Flash (On et al. 2012; Chen 2009) with some concepts (e.g., group commit) being adapted by approaches for NVM. As a consequence, Kim et al. (2016), Gao et al. (2015), and Wang and Johnson (2014) demonstrate how the logging load shifts from I/O-bound to software-bound when utilizing NVM.

In fact, a whole field of persistent programming models for NVRAM as RAM replacement has emerged (Boehm and Chakrabarti 2016; Kolli et al. 2016; Coburn et al. 2011). This includes lightweight logging schemes, (Chatzistergiou et al. 2015b), and approaches for main memory DBMS, DeBrabant et al. (2014b). An alternative approach is KaminoTx, Memaripour et al.

(2017), which avoids copies along the critical I/O path due to logging by permanently maintaining an entire copy of relevant database objects. If a transaction aborts, objects can be recovered from this copy, while updates are handled asynchronously. However, Huang et al. (2014) points out that simply replacing all disks with NVM led to no cost-effective optimum in terms of monetary cost per transaction. As an alternative, the proposed system NV-Logging shows the overheads and scalability limitations of centralized log buffers and therefore enables per-transaction logging.

Liu et al. (2014) first propose the novel unified working memory and persistent store architecture, NVM Duet, which provides the required consistency and durability guarantees. Later, Liu et al. (2017) present DudeTM, a crash-consistent durable transaction system that avoids the drawbacks of both undo and redo logging while providing NVM access directly through CPU load and store instructions.

FOEDUS, Kimura (2015), is a holistic approach addressing OLTP DBMS on novel hardware. FOEDUS exploits OCC and is based on Masstree (Mao et al. 2012) and SILO (Tu et al. 2013). It introduces the concepts of dual pages to exploit NVRAM for large cold data and DRAM for hot data, eliminating the need for mapping tables that reduce the concurrency and scalability. In addition, dual pages are coupled with a technique called stratified snapshots for mirroring.

Recovery and Instant Restart

Since the characteristics of modern storage technologies, especially NVM, can be exploited to reduce the recovery time of data management systems, multiple approaches are proposed. The concept of instant restart is introduced in Oukid et al. (2014, 2015), and Schwalb et al. (2016), while instant recovery is described in Graefe et al. (2015, 2016), Graefe (2015), and Sauer et al. (2015). A novel logging and recovery protocol, write-behind logging, is defined by Arulraj et al. (2016) and represents the key idea of logging *what* parts have changed rather than *how* it was changed and gives the DBMS the opportunity to recover nearly instantaneously from system failures.

Indexing and Index Structures

This research area focuses on hybrid DRAM-NVM settings with the goal of improving write efficiency and utilizing read/write asymmetry and byte addressability. There are numerous proposals for indexing techniques in main memory databases: many are cache-aware, are latch-free, and rely on the symmetry of access. Chen et al. (2011) shows how these are suboptimal for PCM, due to the write operations caused by index updates and instead propose an improved B⁺-Tree algorithm for PCMs. B-Trees on NVM have been explored in Chi et al. (2014), Yang et al. (2015), Chen and Jin (2015), and Chen et al. (2011).

Oukid et al. (2016) with its Fingerprinting Persistent Tree (FPTree) addresses the latency sensitivity issues of current persistent trees approaches for NVM. It only persists leaf nodes on NVM and maintains inner nodes in DRAM, rebuilding them upon recovery. In this manner, the applied technique Fingerprinting limits the expected number of in-leaf probed keys to one. Write optimizations are achieved by keeping leaf node entries unsorted, yielding less cache line writes and lower write amplification. Viglas (2012) and Ben-David et al. (2016) address read/write asymmetry by relaxing the B+Tree balance.

Apart from hybrid DRAM-NVM settings, Sadoghi et al. (2016) demonstrate significant reductions in index maintenance, query processing, and insertion time for multiversion databases by redesigning persistence index structures. Employing an extra level of indirection, stored on Flash storage, the number of magnetic disk I/Os for previous metrics can be reduced dramatically.

Examples of Application

Examples for fully mature DBMSs with focus on transaction processing on NVM include:

Peloton: With the main focus on the autonomous operation of the DBMS, Pavlo et al. (2017) present Peloton. It is designed for HTAP workloads and builds its current architecture on in-memory database concepts. In the view of the authors, these concepts endure even with the introduction of NVM.

SAP HANA: An early adoption of NVM within the SAP HANA database is presented in Mihnea et al. (2017). Here, the architectural choices are explained, and arising challenges with the NVM integration are discussed.

Cross-References

- ▶ Emerging Hardware Technologies
- ▶ In-Memory Transactions
- ▶ Storage Hierarchies for Big Data
- ▶ Storage Technologies for Big Data
- ▶ Weaker Consistency Models/Eventual Consistency

References

- Arulraj J, Pavlo A (2017) How to build a non-volatile memory database management system. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD'17. ACM, New York, pp 1753–1758. <https://doi.org/10.1145/3035918.3054780>
- Arulraj J, Pavlo A, Dulloor SR (2015) Let's talk about storage & recovery methods for non-volatile memory database systems. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD'15. ACM, New York, pp 707–722. <https://doi.org/10.1145/2723372.2749441>
- Arulraj J, Perron M, Pavlo A (2016) Write-behind logging. Proc VLDB Endow 10(4):337–348. <https://doi.org/10.14778/3025111.3025116>
- Ben-David N, Blelloch GE, Fineman JT, Gibbons PB, Gu Y, McGuffey C, Shun J (2016) Parallel algorithms for asymmetric read-write costs. In: Proceedings of the 28th ACM symposium on parallelism in algorithms and architectures, SPAA'16. ACM, New York, pp 145–156. <https://doi.org/10.1145/2935764.2935767>
- Boehm HJ, Chakrabarti DR (2016) Persistence programming models for non-volatile memory. In: Proceedings of ISMM
- Chatzistergiou A, Cintra M, Viglas SD (2015a) Rewind: recovery write-ahead system for in-memory non-volatile data-structures. Proc VLDB Endow 8(5): 497–508. <https://doi.org/10.14778/2735479.2735483>
- Chatzistergiou A, Cintra M, Viglas SD (2015b) Rewind: recovery write-ahead system for in-memory non-volatile data-structures. Proc VLDB Endow 8(5): 497–508
- Chen S (2009) Flashlogging: exploiting flash devices for synchronous logging performance. In: SIGMOD
- Chen S, Jin Q (2015) Persistent b+-trees in non-volatile main memory. Proc VLDB Endow 8(7):786–797
- Chen S, Gibbons PB, Nath S (2011) Rethinking database algorithms for phase change memory. In: Proceedings of CIDR

- Chi P, Lee WC, Xie Y (2014) Making b+-tree efficient in pcm-based main memory. In: Proceedings of ISLPED, vol 454, pp 69–74
- Coburn J, Caulfield AM, Akel A, Grupp LM, Gupta RK, Jhala R, Swanson S (2011) Nv-heaps: making persistent objects fast and safe with next-generation, non-volatile memories. SIGPLAN Not 46(3):105–118
- Condit J, Nightingale EB, Frost C, Ipek E, Lee B, Burger D, Coetzee D (2009) Better I/O through byte-addressable, persistent memory. In: Proceedings of SOSP
- DeBrabant J, Arulraj J, Pavlo A, Stonebraker M, Zdonik SB, Dulloor S (2014a) A prolegomenon on OLTP database systems for non-volatile memory. In: International workshop on accelerating data management systems using modern processor and storage architectures – ADMS’14, Hangzhou, 1 Sept 2014, pp 57–63
- DeBrabant J, Arulraj J, Pavlo A, Stonebraker M, Zdonik SB, Dulloor S (2014b) A prolegomenon on OLTP database systems for non-volatile memory. In: Proceedings of ADMS
- Dulloor SR, Kumar S, Keshavamurthy A, Lantz P, Reddy D, Sankaran R, Jackson J (2014) System software for persistent memory. In: Proceedings of EuroSys, pp 15:1–15:15
- Gao S, Xu J, Harder T, He B, Choi B, Hu H (2015) Pcm-logging: optimizing transaction logging and recovery performance with PCM. TKDE 27(12):3332–3346
- Graefe G (2015) Instant recovery for data center savings. SIGMOD Rec 44(2):29–34
- Graefe G, Sauer C, Guy W, Härdter T (2015) Instant recovery with write-ahead logging. Datenbank-Spektrum 15(3):235–239
- Graefe G, Guy W, Sauer C (2016) Instant recovery with write-ahead logging: page repair, system restart, media restore, and system failover. Synthesis lectures on data management, 2nd edn. Morgan & Claypool Publishers, San Rafael
- Haddock S, Petrov I, Gottstein R, Buchmann A (2017) From in-place updates to in-place appends: revisiting out-of-place updates on flash. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD’17. ACM, New York, pp 1571–1586. <https://doi.org/10.1145/3035918.3035958>
- Huang J, Schwan K, Qureshi MK (2014) NVRAM-aware logging in transaction systems. Proc VLDB Endow 8(4):389–400. <https://doi.org/10.14778/2735496.2735502>
- Intel (2017) libpmem: persistent memory programming. <http://pmem.io>
- JEDEC (2017) JEDEC DDR5 and NVDIMM-P standards under development. <https://www.jedec.org/news/pressreleases/jedec-ddr5-nvdimm-p-standards-under-development>
- Kim YR, Whang KY, Song IY (2010) Page-differential logging: an efficient and DBMS-independent approach for storing data into flash memory. In: Proceedings of SIGMOD, pp 363–374
- Kim K, Lee S, Moon B, Park C, Hwang JY (2011) IPL-P: in-page logging with PCRAM. PVLDB 4(12): 1363–1366
- Kim WH, Kim J, Baek W, Nam B, Won Y (2016) NVWAL: exploiting NVRAM in write-ahead logging. SIGOPS Oper Syst Rev 50(2):385–398
- Kimura H (2015) Foedus: OLTP engine for a thousand cores and NVRAM. In: Proceedings of SIGMOD
- Kolli A, Pelley S, Saidi A, Chen PM, Wenisch TF (2016) High-performance transactions for persistent memories. SIGPLAN Not 51(4):399–411
- Lee S, Moon B, Park C, Hwang JY, Kim K (2010a) Accelerating in-page logging with non-volatile memory. IEEE Data Eng Bull 33(4):41–47
- Lee S, Moon B, Park C, Hwang JY, Kim K (2010b) Accelerating in-page logging with non-volatile memory. IEEE Data Eng Bull 33(4):41–47
- Liu RS, Shen DY, Yang CL, Yu SC, Wang CYM (2014) NVM duet. In: ASPLOS. ACM Press, New York, pp 455–470. <https://doi.org/10.1145/2541940.2541957>, <http://dl.acm.org/citation.cfm?id=2541940.2541957>
- Liu M, Zhang M, Chen K, Qian X, Wu Y, Zheng W, Ren J (2017) DudeTM. In: ASPLOS. ACM Press, New York, pp 329–343. <https://doi.org/10.1145/3037697.3037714>, <http://alchem.usc.edu/portal/static/download/dudetm.pdf>, <http://dl.acm.org/citation.cfm?doid=3037697.3037714>
- Ma L, Arulraj J, Zhao S, Pavlo A, Dulloor SR, Giardino MJ, Parkhurst J, Gardner JL, Doshi K, Zdonik S (2016) Larger-than-memory data management on modern storage hardware for in-memory OLTP database systems. In: Proceedings of the 12th international workshop on data management on new hardware, DaMoN’16. ACM, New York, pp 9:1–9:7. <https://doi.org/10.1145/2933349.2933358>
- Mao Y, Kohler E, Morris RT (2012) Cache craftiness for fast multicore key-value storage. In: Proceedings of EuroSys
- Memaripour A, Badam A, Phanishayee A, Zhou Y, Alagappan R, Strauss K, Swanson S (2017) Atomic in-place updates for non-volatile main memories with kamino-Tx. In: EuroSys, pp 499–512. <https://doi.org/10.1145/3064176.3064215>, <http://dl.acm.org/citation.cfm?doid=3064176.3064215>
- Mihnea A, Lemke C, Radestock G, Schulze R, Thiel C, Blanco R, Meghlan A, Sharique M, Seifert S, Vishnoi S, Booss D, Peh T, Schreter I, Thesing W, Wagle M, Willhalm T (2017) Sap hana adoption of non-volatile memory. Proc VLDB Endow 10(12):1754–1765. <https://doi.org/10.14778/3137765.3137780>
- Ogleari MA, Miller EL, Zhao J (2016) Relaxing persistent memory constraints with hardware-driven undo+redo logging. STABLE Tech Rep 002:1–18. https://pdfs.semanticscholar.org/0910/14f8f886b88d60d981816c12665ec4667672.pdf?_ga=2.32218686.182960926.1513867920-1003521937.1513867920

- On ST, Xu J, Choi B, Hu H, He B (2012) Flag commit: supporting efficient transaction recovery in flash-based dbmss. *IEEE Trans Knowl Data Eng* 24(9):1624–1639
- Oukid I, Booss D, Lespinasse A, Lehner W, Willhalm T, Gomes G (2017) Memory management techniques for large-scale persistent-main-memory systems. *Proc VLDB Endow* 10(11):1166–1177. <https://doi.org/10.14778/3137628.3137629>
- Oukid I, Lehner W (2017) Data structure engineering for byte-addressable non-volatile memory. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD’17. ACM, New York, pp 1759–1764. <https://doi.org/10.1145/3035918.3054777>
- Oukid I, Booss D, Lehner W, Bumbulis P, Willhalm T (2014) SOFORT. In: Proceedings of the tenth international workshop on data management on new hardware – DaMoN’14. ACM Press, New York, pp 1–7. <https://doi.org/10.1145/2619228.2619236>, <http://15721.courses.cs.cmu.edu/spring2016/papers/a8-oukid.pdf>, <http://dl.acm.org/citation.cfm?doid=2619228.2619236>
- Oukid I, Lehner W, Kissinger T, Willhalm T, Bumbulis P (2015) Instant recovery for main memory databases. In: Proceedings of CIDR
- Oukid I, Lasperas J, Nica A, Willhalm T, Lehner W (2016) FPTree: a hybrid SCM-dram persistent and concurrent b-tree for storage class memory. In: Proceedings of the 2016 international conference on management of data, SIGMOD’16. ACM, New York, pp 371–386. <https://doi.org/10.1145/2882903.2915251>
- Pavlo A, Angulo G, Arulraj J, Lin H, Lin J, Ma L, Menon P, Mowry TC, Perron M, Quah I, Santurkar S, Tomasic A, Toor S, Aken DV, Wang Z, Wu Y, Xian R, Zhang T (2017) Self-driving database management systems. In: CIDR. <http://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf>
- Pelley S, Wenisch TF, Gold BT, Bridge B (2013a) Storage management in the NVRAM era. *Proc VLDB Endow* 7(2):121–132. <https://doi.org/10.14778/2732228.2732231>
- Pelley S, Wenisch TF, Gold BT, Bridge B (2013b) Storage management in the NVRAM era. *Proc VLDB Endow* 7(2):121–132
- Sadoghi M, Ross KA, Canim M, Bhattacharjee B (2016) Exploiting SSDs in operational multiversion databases. *VLDB J* 25(5):651–672
- Sauer C, Graefe G, Härdter T (2015) Single-pass restore after a media failure. In: Proceedings of BTW, pp 217–236
- Schwalb D, Berning T, Faust M, Dreseler M, Plattner H (2015) NVM malloc: memory allocation for NVRAM. In: Proceedings of ADMS, pp 61–72
- Schwalb D, Faust M, Dreseler M, Flemming P, Plattner H (2016) Leveraging non-volatile memory for instant restarts of in-memory database systems. In: 2016 IEEE 32nd international conference on data engineering (ICDE), pp 1386–1389. <https://doi.org/10.1109/ICDE.2016.7498351>
- Tu S, Zheng W, Kohler E, Liskov B, Madden S (2013) Speedy transactions in multicore in-memory databases. In: Proceedings of SOSP, pp 18–32
- Viglas SD (2012) Adapting the B-tree for asymmetric I/O. In: Proceedings of the 16th East European conference on advances in databases and information systems, ADBIS’12. Springer, Berlin/Heidelberg, pp 399–412. https://doi.org/10.1007/978-3-642-33074-2_30
- Viglas SD (2015) Data management in non-volatile memory. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD’15. ACM, New York, pp 1707–1711. <https://doi.org/10.1145/2723372.2731082>
- Volos H, Tack AJ, Swift MM (2011) Mnemosyne: lightweight persistent memory. *Asplos* 22:1–13. <https://doi.org/10.1145/1950365.1950379>, http://research.cs.wisc.edu/sonar/papers/mnemosyne-osdi2010-poster_abstract.pdf
- Volos H, Nalli S, Panneerselvam S, Varadarajan V, Saxena P, Swift MM (2014) Aerie: flexible file-system interfaces to storage-class memory. In: Proceedings of EuroSys, pp 14:1–14:14
- Wang T, Johnson R (2014) Scalable logging through emerging non-volatile memory. *Proc VLDB Endow* 7(10):865–876
- Xu J, Swanson S (2016) Nova: a log-structured file system for hybrid volatile/non-volatile main memories. In: Proceedings of FAST
- Yang J, Wei Q, Chen C, Wang C, Yong KL, He B (2015) Nv-tree: reducing consistency cost for NVM-based single level systems. In: Proceedings of FAST, pp 167–181

HDFS

► Hadoop

Healthcare

► Big Data for Health

Healthcare Ontologies

► Big Semantic Data Processing in the Life Sciences Domain

Hierarchical Process Discovery

Raffaele Conforti

University of Melbourne, Melbourne, VIC,
Australia

Synonyms

Automated discovery of hierarchical process models

Definitions

Hierarchical process discovery is a family of methods in process mining that starting from an event log focuses on the automated discovery of process models containing one or more subprocesses, also known as hierarchical process models.

Overview

With the increasing availability of business process execution data, i.e., event logs, the use of automated process discovery techniques is becoming a common practice among business process management practitioners as a mean to quickly gain insights about the execution of a business process.

Despite several automated process discovery techniques had been proposed over the years (van der Aalst et al. 2004; Weijters and Ribeiro 2011; Leemans et al. 2013), these techniques fall short when dealing with event logs of processes containing sub-processes. When dealing with this type of event logs, classical automated process discovery often produces flat process models that are highly complex and inaccurate (Li et al. 2011; Weber et al. 2015; Sun and Bauer 2016). Let us consider, for example, the log of a process containing a sub-process that may generate multiple instances running at the same time within the same process instance. An example of the application of classical automated process discovery (Weijters and Ribeiro 2011)

on top of this event log is illustrated in Fig. 1. In this particular case, classical automated process discovery fails to isolate the behavior of the sub-process, resulting in a complex model with a high number of gateways and arcs. To overcome this limitation, hierarchical process discovery techniques which reduce the complexity of a discovered process model by exploiting the use of sub-processes have been proposed.

Hierarchical Process Discovery

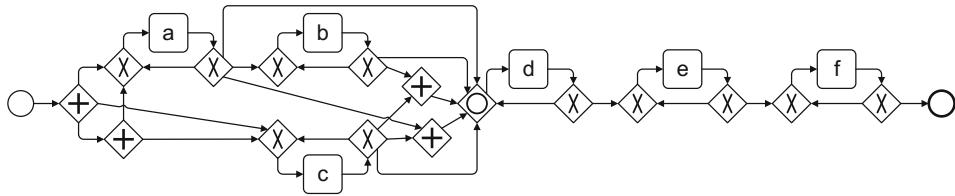
Since the proposal of the first hierarchical process discovery technique by Li et al. (2011), many other techniques followed Maggi et al. (2014), Sun and Bauer (2016), Conforti et al. (2014), Conforti et al. (2016a,b), Wang et al. (2015), and Weber et al. (2015). Despite all these techniques are based on different ideas, the discovery of a hierarchical process model is generally achieved through a two-phase approach (see Fig. 2). Starting from an event log, the first phase involves the extraction of sub-logs from the event log, which leads to a hierarchy of logs with one parent event log and several separate sub-logs (one for each sub-process). In the second phase, a hierarchical process model is discovered by applying classical automated process discovery on top of each (sub-)log.

Sub-logs Extraction

When considering the approach used for the identification and extraction of the hierarchy of logs, hierarchical process discovery techniques can be divided in two major groups: (i) techniques relying on control flow information and (ii) techniques relying on data attributes.

Control-Flow-Based Techniques

Control flow-based techniques are those techniques that identify a sub-log relying uniquely on information that can be derived from the order of the events within an event log. Hierarchical process discovery techniques belonging to this category are the ones proposed by Li et al. (2011), Maggi et al. (2014), and Sun and Bauer (2016).



Hierarchical Process Discovery, Fig. 1 Flat process model (Conforti et al. 2014, 2016b)

Hierarchical Process Discovery, Fig. 2 The two-phase approach



The technique proposed by Li et al. (2011) is based on the idea that the execution of a sub-process leaves a specific *footprint* in an event log. Such a footprint takes the shape of a recurrent execution pattern (Bose and van der Aalst 2009), which can then be used to detect the presence of a sub-process. To detect a sub-process, the authors rely on two patterns: (i) the tandem arrays pattern that corresponds to the execution of loops and (ii) the maximal repeats pattern that corresponds to maximal common subsequence of activities across several process instances. Using these two patterns, sequences of events which could be group together within the same sub-log are identified and several sub-logs extracted. To limit the number of sub-logs generated, only the most frequent patterns out of all patterns detected are considered.

Maggi et al. (2014) consider sub-processes as *pockets of flexibility* within which events of an event log may either be structured (i.e., distributed as a sequence for which the following event can be easily predicted) or unstructured. This concept of structured and unstructured events is used to split the event log in sub-logs. This is achieved by grouping within the same sub-log all subsequent structured events, while unstructured events are grouped together when they are situated in between sequences of structured events.

Finally, the technique proposed by Sun and Bauer (2016) is based on the idea that activities belonging to the same sub-process will be densely connected and their events will

frequently follow each other within the event log. Following this concept, the authors propose to detect densely connected activities by employing graph clustering techniques on top of a causal activity graph, i.e., a graph depicting the causal relation between activities. The resulting clusters are then used to generate several sub-logs (one for each cluster), where each sub-log contains all events belonging to activities within the same cluster.

Data-Based Techniques

In contrast to control flow-based techniques, data-based techniques rely on data attributes belonging to events for the identification of sub-logs and their hierarchical structure. Techniques belonging to this category are the ones proposed by Conforti et al. (2014, 2016b), Wang et al. (2015), and Weber et al. (2015).

The technique proposed by Conforti et al. (2014, 2016a,b) identifies a hierarchy of sub-logs within the event log by relying on approximate functional and inclusion dependency discovery techniques (Huhtala et al. 1999; Bauckmann et al. 2010; Zhang et al. 2010). Additionally, using approximate dependency discovery techniques (Zhang et al. 2010), the technique can identify and filter out noise from the event log, which may be caused by data entry errors, missing event records, or infrequent behavior (Conforti et al. 2016b).

Similarly to Conforti et al. (2014, 2016a,b), the technique proposed by Weber et al. (2015) is based on the idea that a hierarchy between

sub-logs can be identified using data attributes belonging to events. In contrast with the approach by Conforti et al. (2014, 2016a,b), Weber et al. (2015) do not propose a technique for the identification of such data attributes but assume them to be provided in input.

In contrast with these two techniques, the availability of data attributes that can be used to identify a hierarchy between sub-logs is a concern for Wang et al. (2015). To overcome this limitation, they propose a technique that relies on the start and end execution time of each event. Using start and end execution times, they generate an instance tree for each process instance recorded in the log. An instance tree is a tree where events which may occur in parallel are placed on the same level of the hierarchy. These instance trees are then merged together in a hierarchy tree, and events sharing the same ancestor and the same descendants over the union of all instance trees are grouped together within the same sub-log.

The concept of sub-logs extraction is not limited to hierarchical process discovery, and it can be found both in trace clustering as well as in decomposed process discovery.

Trace clustering techniques divide the set of traces of an event log into multiple clusters, where each cluster contains a set of similar traces. These traces correspond to the traces of a variant of a process. The main difference from hierarchical process discovery is that, while in hierarchical process discovery the extracted sub-logs consist of partial traces, in the context of trace clustering, each of the resulting clusters consists of a set of full traces. Additionally, in hierarchical process discovery, a trace at a higher level may refer to a sub-trace at a lower level in the hierarchy.

Similarly, for decomposed process discovery, the goal is to break down each trace in the log into multiple sub-traces. However, in decomposed process discovery, all the sub-traces are at the same level. Moreover, the purpose is different: in hierarchical process discovery, the goal is to discover a modular process model, whereas in decomposed process mining, the goal is to discover a flat process model in a scalable manner.

Sub-process Discovery

When considering the discovery of each sub-process model, all hierarchical process discovery techniques discussed so far rely on classical automated process discovery techniques. Despite this commonality, hierarchical process discovery techniques differ in terms of the features that a discovered process model may contain.

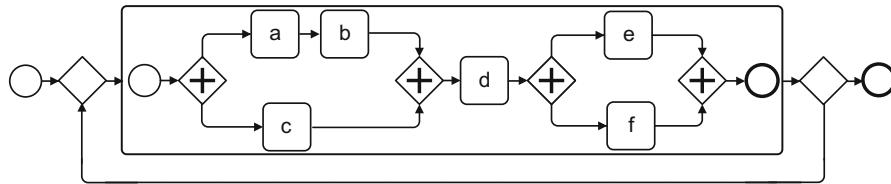
In the following, we provide a comparison of each technique using the set of constructs supported by Business Process Model and Notation (BPMN) (Object Management Group (OMG) 2011) as a reference. When considering hierarchical process discovery and the type of constructs which may be automatically included into a sub-process, we can subdivide these techniques in three groups: (i) techniques supporting the core set of BPMN constructs, (ii) techniques supporting the core set of BPMN constructs and activity markers, and (iii) techniques supporting the complete set of BPMN constructs.

Core BPMN

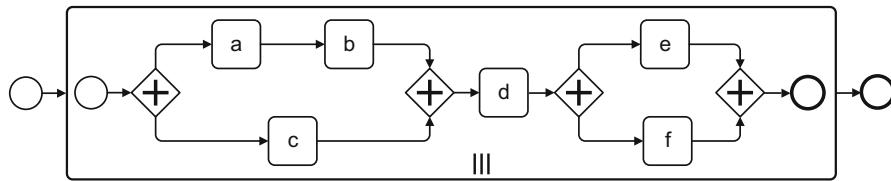
In this category, we find techniques discovering sub-processes which only contain the core set of BPMN constructs, such as start and end events, activities, and gateways. Techniques falling into this category are the ones that outsource the discovery of a sub-process to classical automated process discovery techniques without performing any additional post-processing operation. Models discovered by techniques in this category present a structure similar to the one illustrated in Fig. 3. In comparison to the model presented in Fig. 1, these techniques are able to identify the presence of a sub-process resulting in a simpler process model, despite the model will only allow the sub-process to be executed in a loop (see gateways located externally to the sub-process). In this category, we find the techniques proposed by Li et al. (2011), Maggi et al. (2014), and Sun and Bauer (2016).

Core BPMN and Activity Markers

This category encompasses techniques discovering sub-processes which make use of activity markers in addition to the core set of BPMN constructs. Activity markers are used to

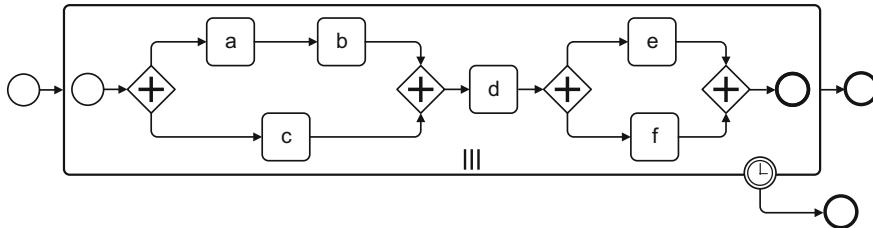


Hierarchical Process Discovery, Fig. 3 Hierarchical process model – core BPMN



H

Hierarchical Process Discovery, Fig. 4 Hierarchical process model – core BPMN with activity markers



Hierarchical Process Discovery, Fig. 5 Hierarchical process model – complete BPMN

differentiate sub-processes which may be executed in a loop or for which several instances of the same sub-process may be spawned at the same time. Techniques falling into this category detect if a sub-process requires an activity marker by using heuristics which are invoked after the discovery of each sub-process. These heuristics generally check if multiple instances of a sub-process are occurring within the same process instance. If this is the case and they overlap in time, the sub-process is marked as multi-instance; otherwise it is marked as loop. In this category, we find the techniques by Wang et al. (2015) and Weber et al. (2015). Process models produced by these techniques will resemble the one illustrated in Fig. 4. In this case, for example, these techniques detect that multiple instances of the sub-process are executed concurrently; hence the use of the multi-instance activity marker in the sub-process replaces the gateways located externally to the sub-process originally used to allow the sub-process to be repeated (Fig. 5).

Complete BPMN

Finally, this category contains techniques able to discover sub-processes that make use of the entire set of BPMN constructs. In this category, we find the technique by Conforti et al. (2014, 2016a,b). In addition to the ability to identify sub-processes that are executed in a loop or that may spawn multiple instances (achieved using similar heuristics as the ones adopted by the techniques presented in the second category), this technique can detect if a given sub-process is subjected to a special condition which forces the premature termination of the sub-process. To achieve this, the technique checks whether the last activity of the sub-process is always executed (i.e., it is present in each instance of the sub-log). The absence of this activity from one of the instances is considered as a sign for a premature termination. Whenever a premature termination is detected, a boundary event is attached to the sub-process. The nature of the premature termination will then determine the type of boundary event,

Hierarchical Process Discovery, Table 1 Hierarchical process discovery techniques: a summary

Technique	Sub-logs extraction based on		Sub-process discovery		
	Control-flow	Data	Core	Activity markers	Complete
Li et al. (2011)	✓		✓		
Maggi et al. (2014)	✓		✓		
Sun and Bauer (2016)	✓		✓		
Wang et al. (2015)		✓		✓	
Weber et al. (2015)		✓		✓	
Conforti et al. (2014, 2016a,b)		✓			✓

i.e., timer event or message event. Specifically, a timer boundary event is used when the duration of a sub-process never exceeds a certain amount of time, while a message boundary event is used in all remaining cases.

Table 1 provides a brief summary of all techniques presented, with the relative subcategory classification.

Conclusion and Final Considerations

In previous sections, we discussed the importance of hierarchical process discovery, we highlighted the advantages that it has over classical automated process discovery, and we presented several techniques that can be used to discover hierarchical process models. In this final section, we will discuss some aspects that should be considered when undertaking the discovery of a hierarchical process model.

Hierarchical process discovery and the quality of a discovered hierarchical process model are influenced by two main factors: the quality of the event log and the effectiveness of the classical automated process discovery technique used for the discovery of each sub-process.

The quality of an event log is a deciding factor over the type of hierarchical process discovery technique that should be adopted. In particular, if data attributes are missing or if the quality of information recorded is inadequate, data-based techniques will fail to identify possible sub-logs within the event log, which in turn will result into the discovery of a flat process model.

Similarly, the quality of a discovered hierarchical process model is affected by the effectiveness of the classical automated process discovery technique used for the discovery of each sub-process. In particular, techniques performing poorly will produce imprecise and complex subprocesses which will still result into imprecise and complex hierarchical process models.

These two aspects should be taken into consideration when attempting to gain insights about a business process. Business process management practitioners should consider hierarchical process discovery techniques only when the quality of the event log is adequate. In scenarios where logs are of poor quality (e.g., logs containing incorrectly recorded events and where data attributes are unreliable or missing), control flow-based hierarchical process discovery techniques should be preferred over data-based hierarchical process discovery techniques. Finally, the discovery of each sub-process should be handled through the use of robust classical automated process discovery techniques for which we refer the reader to the entry on automated process discovery, where a detailed analysis of all available techniques is offered.

Finally, the problem of hierarchical process discovery shares several similarities with the problem of artifact-centric process discovery. Both families of methods aim at discovering a modular process model from an event log. However, they differ in terms of their modularization approach. Specifically, artifact-centric process discovery differs from hierarchical process discovery as it seeks to discover a process model

that consists of a set of interacting artifacts, each one with an independent life cycle. For a detailed discussion on artifact-centric process discovery, we refer to the corresponding entry in this encyclopedia.

Cross-References

- ▶ [Artifact-Centric Process Mining](#)
- ▶ [Automated Process Discovery](#)
- ▶ [Decomposed Process Discovery and Conformance Checking](#)

References

- Bauckmann J, Leser U, Naumann F (2010) Efficient and exact computation of inclusion dependencies for data integration. Technical report 34, Hasso-Plattner-Institute
- Bose RPJC, van der Aalst WMP (2009) Abstractions in process mining: a taxonomy of patterns. In: Proceedings of the 7th international conference on business process management. Lecture notes in computer science, vol 5701. Springer, pp 159–175
- Conforti R, Dumas M, García-Bañuelos L, La Rosa M (2014) Beyond tasks and gateways: discovering BPMN models with subprocesses, boundary events and activity markers. In: Proceedings of the 12th international conference on business process management. Lecture notes in computer science, vol 8659. Springer, pp 101–117
- Conforti R, Augusto A, Rosa ML, Dumas M, García-Bañuelos L (2016a) BPMN miner 2.0: discovering hierarchical and block-structured BPMN process models. In: Proceedings of the BPM demo track 2016 co-located with the 14th international conference on business process management, CEUR-WS.org, CEUR workshop proceedings, vol 1789, pp 39–43
- Conforti R, Dumas M, García-Bañuelos L, La Rosa M (2016b) BPMN miner: automated discovery of BPMN process models with hierarchical structure. Inf Syst 56:284–303
- Huhtala Y, Kärkkäinen J, Porkka P, Toivonen H (1999) TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput J 42(2):100–111
- Leemans SJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs – a constructive approach. In: Proceedings of the 34th international conference on application and theory of petri nets and concurrency. Lecture notes in business information processing, vol 7927. Springer, pp 311–329
- Li J, Bose RPJC, van der Aalst WMP (2011) Mining context-dependent and interactive business process maps using execution patterns. In: Proceedings of business process management workshops. Lecture notes in business information processing, vol 66. Springer, pp 109–121
- Maggi FM, Slaats T, Reijers HA (2014) The automated discovery of hybrid processes. In: Proceedings of the 12th international conference on business process management. Lecture notes in computer science, vol 8659. Springer, pp 392–399
- Object Management Group (OMG) (2011) Business process model and notation (BPMN) ver. 2.0. Object Management Group (OMG). <http://www.omg.org/spec/BPMN/2.0>
- Sun Y, Bauer B (2016) A graph and trace clustering-based approach for abstracting mined business process models. In: Proceedings of the 18th international conference on enterprise information systems, SciTePress, pp 63–74
- van der Aalst WMP, Weijters T, Maruster L (2004) Workflow mining: discovering process models from event logs. IEEE Trans Knowl Data Eng 16(9):1128–1142
- Wang Y, Wen L, Yan Z, Sun B, Wang J (2015) Discovering BPMN models with sub-processes and multi-instance markers. In: Proceedings of the on the move (OTM) confederated international conferences. Lecture notes in computer science, vol 9415. Springer, pp 185–201
- Weber I, Farshchi M, Mendling J, Schneider J (2015) Mining processes with multi-instantiation. In: Proceedings of the 30th annual ACM symposium on applied computing. ACM, pp 1231–1237
- Weijters AJMM, Ribeiro JTS (2011) Flexible heuristics miner (FHM). In: Proceedings of the IEEE symposium on computational intelligence and data mining. IEEE, pp 310–317
- Zhang M, Hadjieleftheriou M, Ooi BC, Procopiuc CM, Srivastava D (2010) On multi-column foreign key discovery. Proc VLDB Endow 3(1):805–814

H

Historical Graph Management

Udayan Khurana¹ and Amol Deshpande²

¹IBM Research AI, TJ Watson Research Center, New York, NY, USA

²Computer Science Department, University of Maryland, College Park, MD, USA

Definitions

Real-world graphs evolve over time, with continuous addition and removal of vertices and

edges, as well as frequent change in their attribute values. For decades, the work in graph analytics was restricted to a static perspective of the graph. In recent years, however, we have witnessed an increasing abundance of timestamped observational data, fueling an interest in performing richer analysis of graphs, along a temporal dimension. However, the traditional graph data management systems that were designed for static graphs provide inadequate support for such temporal analyses. We present a summary of recent advances in the field of historical graph data management. They involve, compact storage of large graph histories, efficient retrieval of temporal subgraphs, and effective interfaces for expressing historical graph queries, essential for enabling temporal graph analytics.

Overview

There is an increasing availability of timestamped graph data – from social and communication networks to financial networks, biological networks, and so on. Analysis of the history of a graph presents valuable insights into the underlying phenomena that produced the graph. For instance, the evolution of a graph or community can be seen through the evolution of metrics such as diameter or density; change in the importance of key entities in a network can be done through measuring change in network centrality and so on.

Traditional graph data management systems based on static graphs provide insufficient support for temporal analysis over historical graphs. This is due to a fundamental lack in the modeling of the *change* of information. If performed using the conventional graph systems, such tasks become prohibitively expensive in storage costs, memory requirements, or execution time and are often unfriendly or infeasible for an analyst to even express in the first place. The frequent change of a temporal graph also poses a significant challenge to algorithm design, because the overwhelming majority of graph algorithms assume static graph structures. One would have to design special algorithms for each application

to accommodate the dynamic aspects of graphs. To support general-purpose computations, most of the emerging temporal graph systems adopt a strategy to separate graph updates from graph computation. More specifically, although updates are continually applied to a temporal graph, graph computation is only performed on a sequence of successive *static* views of the temporal graph. For simplicity, most systems adopt a *discretized-time* approach, so that time domain is set of natural numbers, i.e., $t \in \mathcal{N}$. As per the terminology of temporal relational databases, this discussion considers *valid time* (against *transaction time*) as the underlying temporal dimension for historical analyses. Valid time denotes the time period during which a fact is true with respect to the real world. Transaction time is the time when a fact is stored in the database. It is worth noting that there is a related but orthogonal body of work which we do not touch in this chapter. It deals with the need to do *real-time analytics on the streaming data* as it is being generated; here, the scope of the analysis typically only includes the latest snapshot or the snapshots from a recent window. The key challenge there is to be able to deal with the high rate at which the data is often generated.

Broadly speaking, this chapter's focus is to briefly introduce the reader to the recent advances in historical graph data management for temporal graph analytics. There are many different types of analyses that may be of interest. For example, an analyst may wish to study the evolution of well-studied static graph properties such as centrality measures, density, conductance, etc. over time or the search and discovery of temporal patterns, where the events that constitute the pattern are spread out over time. Comparative analysis, such as juxtaposition of a statistic over time or, perhaps, computing aggregates such as *max* or *mean* over time, gives another style of knowledge discovery into temporal graphs. Most of all, even a primitive notion of simply being able to access past states of the graphs and performing simple static graph analytics empowers a data scientist with the capacity to perform analysis in arbitrary and unconventional patterns. Supporting such a diverse set of temporal analytics

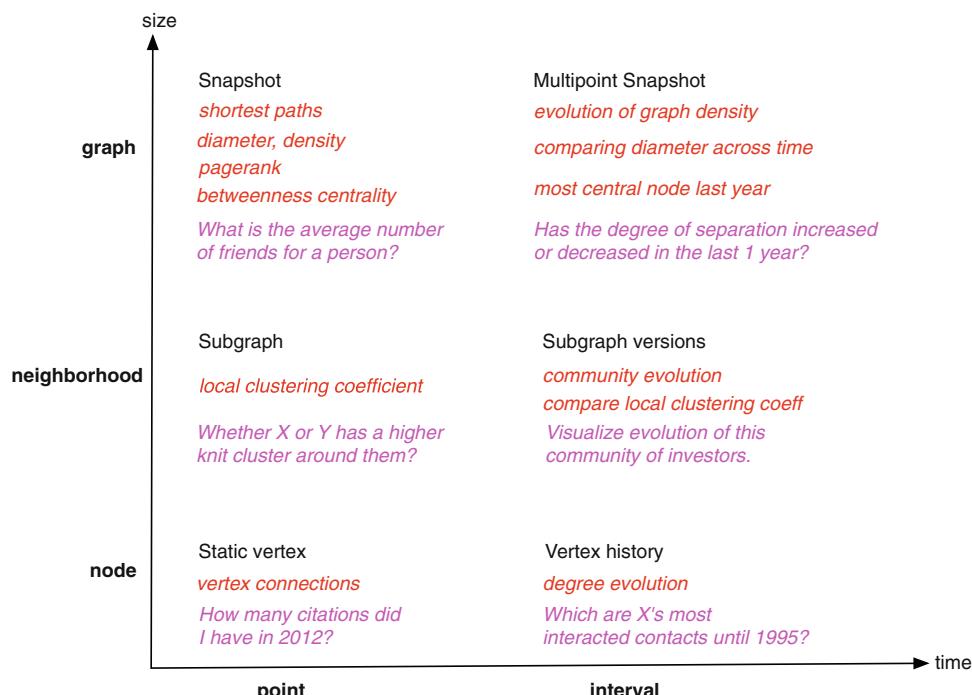
and querying over large volumes of historical graph data requires addressing several data management challenges. Specifically, we need techniques for storing the historical information in a compact manner, while allowing a user to retrieve graph snapshots as of any time point in the past or the evolution history of a specific node or a specific neighborhood. Further the data must be stored and queried in a distributed fashion to handle the increasing scale of the data.

Historical or temporal analyses on graphs span a variety of tasks that vary in the analytical quantity of interest, as well as the scope of the graph locality and time duration being analyzed. Figure 1 provides a classification on those lines and lists a few examples of graph retrieval and analysis tasks. A more exhaustive taxonomy of temporal tasks is provided by Ahn et al. (2014). Finally, there is a need for an expressive, high-level, easy-to-use programming framework that will allow users to specify complex temporal graph analysis tasks in a data-parallel fashion across a cluster.

Key Research Findings

Storage and Retrieval

Storage of large temporal graphs is challenging and requires careful design. An effective storage system for temporal graphs has a twofold objective. First, the storage should be compact such that the invariant information across multiple graph versions is not stored multiple times. Second, it must allow for efficient retrieval of graph primitives such as snapshot(s), temporal range queries of nodes, or neighborhoods, among others, as reflected in Fig. 1. A straightforward use of static graph data stores can lead to an explosion in storage requirements and/or incur high fetch latency times. Consider two basic and extreme approaches from conventional storage methods to support snapshot retrieval queries, referred to as the *Copy* and *Log* approaches, respectively, as described by Salzberg and Tsotras (1999). While the copy approach relies on storing new copies of a snapshot upon every point of change, the log approach relies on storing everything through



Historical Graph Management, Fig. 1 A temporal graph can be represented across two different dimensions – time and entity. It lists retrieval tasks (black), graph

operations (red), example queries (magenta) at different granularities of time and entity size (Khurana 2015)

changes. Their hybrid is often referred to as the *Copy+Log* approach. Copy approach is clearly infeasible for frequently changing graphs because of intractable storage needs; the log approach, while storing minimal information, makes it a hard work to reconstruct any snapshot or temporal subset of the graph.

Delta-Based Encodings

The use of “*deltas*”, or graph differences, is a powerful approach to register the changes in a graph over a period of time. If carefully organized, deltas can provide efficient retrieval at low storage costs. The *DeltaGraph* index (Khurana and Deshpande 2013a), for instance, provides highly efficient retrieval of individual snapshots of the historical graph for specific time instances. It organizes the historical graph data in a hierarchical data structure, whose lowest level corresponds to the snapshots of the network over time and whose interior nodes correspond to graphs constructed by “combining” the lower level snapshots in some fashion; the interior nodes are typically not valid snapshots as of any specific time point. Neither the lowest-level graph snapshots nor the graphs corresponding to the interior nodes are actually stored explicitly. Instead, for each edge, a *delta*, i.e., the difference between the two graphs corresponding to its endpoints, is computed, and these deltas are explicitly stored. In addition, the graph corresponding to the root is explicitly stored. Given those, any specific snapshot can be constructed by traversing any path from the root to the node corresponding to the snapshot in the index and by appropriately combining the information present in the deltas. The use of different “combining” functions leads to a different point in the performance-storage trade-off, with *intersection* being the most natural such function. This index structure is especially effective with *multi-snapshot retrieval* queries, which are expected to be common in temporal analysis, as it can share the computation and retrieval of deltas across the multiple snapshots. While it is efficient at snapshot retrieval, it is not suitable for fetching graph primitives such as histories of nodes or neighborhoods over specified periods of time. However, *Temporal Graph Index*

(*TGI*) (Khurana and Deshpande 2016), is an extension of DeltaGraph which partitions deltas in so-called microdeltas, and uses chaining of related microdeltas across horizontal nodes in the DeltaGraph, allows retrieval of different temporal graph primitives including neighborhood versions, node histories, and graph snapshots. Such a microdelta-based design is great for distributed storage and parallel retrieval on a cloud. This allows efficient retrieval of not only entire snapshots but also of individual neighborhoods or temporal histories of individual neighborhoods.

Storing by Time Locality

Another powerful approach to store and process temporal graphs is based on time locality. *Chronos* (Han et al. 2014) targets time-range graph analytics, requiring computation on the sequence of static snapshots of a temporal graph within a time range. An example is the analysis of change in each vertex’s PageRank for a given time range. Obviously, the most straightforward approach of applying computation on each snapshot separately is too expensive. Chronos achieves efficiency by exploiting locality of temporal graphs. It also leverages the time locality to store temporal graphs on disk in a compact way. The layout is organized in snapshot groups. A snapshot group G_{t_1, t_2} contains the state of G in the time range $[t_1, t_2]$, by including a checkpoint of the snapshot of G at t_1 followed by all the updates made till t_2 . The snapshot group is physically stored as edge files and vertex files in time-locality fashion. For example, an edge file begins with an index to each vertex in the snapshot group, followed by segments of vertex data. The segment of a vertex, in turn, first contains a set of edges associated with the vertex at the start time of the snapshot group, followed by all the edge updates to the vertex. A link structure is further introduced to link edge updates related to the same vertex/edge, so that the state of a vertex/edge at a given time t can be efficiently constructed.

Indexing Using Multiversion Arrays

LLAMA (Macko et al. 2015) presents an approach where an evolving graph is modeled as a time

series of graph snapshots, where each batch of incremental updates produces a new graph snapshot. The graph storage is read-optimized, while the update buffer is write-optimized. It augments the compact read-only CSR representation to support mutability and persistence. Specifically, a graph is represented by a single vertex table and multiple edge tables, one per snapshot. The vertex table is organized as a large multiversioned array (LLAMA) that uses a software copy-on-write technique for snapshotting, and the record of each vertex v in the vertex table maintains the necessary information to track v 's adjacency list from the edge tables across snapshots. The array of records is partitioned into equal-sized data pages, and an indirection array is constructed that contains pointers to the data pages. The indirection array fits in L3 cache. To create a new snapshot, the indirection array is copied, with those references to outdated pages replaced by those to the newly modified pages. Thus, we do not need to copy unmodified pages across snapshots. LAMA stores 16 consecutive snapshots of the vertex table in each file, so that disk space can be easily reclaimed from deleted snapshots. The edge table for a snapshot i is organized as a fixed-length array that stores adjacency list fragments consecutively, where each adjacency list fragment contains the edges of a vertex added in snapshot i . An adjacency list fragment of vertex v also stores a continuation record, which points to the next fragment for v , or *null* if there are no more edges. To support edge deletion, each edge table maintains a deletion vector, which is an array that encodes in which snapshot an edge was deleted.

Analytical Frameworks

Running graph analytics requires dealing with two main aspects – the runtime system components and the interfaces to express the analytical task. Let us discuss the essentials of both.

Runtime Environment Aspects

In-Memory Graph Layouts. Like the storage and indexing of temporal graphs, dealing with temporal redundancy is important in the in-memory data structures on which analytics are

executed. For example, when running a graph algorithm or evaluating a metric on various snapshots of a graph, it might be prohibitively expensive to store all the snapshots in the memory at the same time. However, most graph libraries do require plain isolated version of each graph. The most common solution to this problem is to use an overlaying of multiple versions on one another and using bitmaps and an additional lookup table to establish the validity of a graph component to one or more versions. This reduces any redundancy of nodes, edges, or attributes that occur across multiple versions yet obtaining access to each snapshot as a static graph through a simple interface. GraphPool (Khurana and Deshpande 2013a) and Chronos (Han et al. 2014) among others use variations of this technique. In addition, Chronos also employs such a structure for locality-aware batch scheduling (LABS) of graph computation. More specifically, LABS batches the processing of a vertex across all the snapshots, as well as the information propagation to a neighboring vertex for all the snapshots.

Parallelism. Temporal graph analytics presents opportunities for parallelization in two different ways. First, evaluating a property across multiple snapshots can be easily parallelized, unless. Second, several static graph computation operations, such as PageRank, can be run in parallel themselves. Determining the optimum level of parallelism for an analytical task can be complex and depends on the exact computation steps, nature of graph data, and the resources available. TGAF (Khurana and Deshpande 2016) provides parallelism by expressing all temporal graphs as RDDs of time evolving nodes or subgraphs and letting the underlying Spark infrastructure plan the parallelism. Leveraging Spark parallelism provides abundant benefits, along with simplicity of design. However, there are occasions on which certain tasks can be customized for better results. A major drawback with TGAF is the lack of space saving obtained through overlaid structures such as GraphPool. LABS and Llama show effective locality-based multi-core parallelism.

Incremental Computation. Time-range graph analysis can benefit from incremental or shared computation. First, for iterative algorithms such as PageRank, if the target time range contains a sequence of N snapshots S_0 to S_{N-1} , the result computed on S_0 can be used to initialize the solution for S_1 and so on until S_{N-1} . This reduces the number of iterations needed to converge an algorithm but also serializes the computations because of the newly created dependence. Chronos utilizes a variation of this principle to effectively perform incremental computation.

Analytical Interfaces

There are mainly two kinds of historical graph analytical interfaces. The first one, a programmatic interface involves an abstraction of a temporal graph and various operations that can be performed on it, along with calls to several static (sub-)graph routines. The second kind are the visual exploration tools which are usually targeted to more specific types of analytic options but provide an easier means to express the task through GUI interfaces and provide meaningful visualization of the results. One example of a programmatic interface is TGAF (Khurana and Deshpande 2016), which abstracts the temporal graph through two main primitives – *Set of temporal nodes (SoN)* and *Set of temporal subgraphs (SoTS)*. The entire temporal graph or any part of it may be expressed as one of these primitives. A SoN is best illustrated as a three-dimensional array along the axes of time, nodes, and attributes. The system also defines graph manipulation operations such as *timeslicing*, *selection*, and *filtering*; compute operators such as *node compute map*, *temporal node compute map*, *node compute delta*, and among others; it also provides analytical operators such as *compare*, *evolution*, and such. A rich set of operators enables the expression of complex analytical operations. We omit further details here and refer the interested reader to Khurana and Deshpande (2016) and Khurana (2015).

Visualization is an effective means for exploring and analyzing a temporal graph. The following is a rather brief and non-exhaustive reference to techniques and tools that have demonstrated

such capabilities. Using visualization of different snapshots, Toyoda and Kitsuregawa (2005) study events like appearance of pages, relationship between different pages for a historical dataset of Web archive. *Hinge* (Khurana and Deshpande 2013b) provides a visual timeline-based exploration and search for a temporal graph; it additionally provides the capability to search simple subgraph patterns in a time range. NetEvViz (Khurana et al. 2011) extends NodeXL (a popular graph analysis tool) to study evolving networks. It uses an edge color coding scheme for comparative study of network growth. Ahn et al. (2011) focus on different states of graph components in order to visualize temporal dynamics of a social network. Alternate representations to node-link diagrams have been proposed in order to study temporal nature of social networks, such as TimeMatrix (Yi et al. 2010) that uses a matrix based visualization to better capture temporal aggregations and overlays. A temporal evolution analysis technique based on capturing visual consistency across snapshots can be seen in the work of Xu et al. (2011). A survey by Beck et al. (2014) lists several other approaches for visual analysis of dynamic graphs.

Examples of Application

In recent years, we have witnessed several works of historical graph analyses. Such work could benefit from the new research and systems described above for historical graph data management. Few examples of such applications are as follows. Leskovec et al. (2007) and Kumar et al. (2010) have designed analytical models that capture network evolution, with a focus on social networks and the Web. Studies analyze how communities evolve (Tang et al. 2008), identifying key individuals, and locating hidden groups in dynamic networks (Tantipathananandh et al. 2007), also characterizing the complex behavioral patterns of individuals and communities over time (Asur et al. 2009). Biologists are interested in discovering historical events leading to a known state of a biological network (Navlakha and Kingsford 2011). Change in page rank

(Bahmani et al. 2010), change in centrality of vertices, path lengths of vertex pairs, etc. (Pan and Saramäki 2011), shortest paths evolution (Ren et al. 2011), and such, provide useful analytical algorithms for changing graphs.

Future Directions for Research

The topic of historical graph data management is ripe with research opportunities. In spite of the advances in temporal graph querying, there is a considerable scope for designing a more expressive and rigorous algebra and language. The work in graph pattern matching and mining appears to lack a temporal dimension. A related but important topic is that of finding interesting temporal graphs from timestamped datasets such as system logs. In terms of systems research, integrating a historical graph system and a streaming graph system into one presents interesting challenges and opportunities for researchers in the field.

Cross-References

- ▶ [Graph Data Management Systems](#)
- ▶ [Graph OLAP](#)
- ▶ [Graph Representations and Storage](#)
- ▶ [Link Analytics in Graphs](#)
- ▶ [Parallel Graph Processing](#)

References

- Ahn JW, Taieb-Maimon M, Sopan A, Plaisant C, Shneiderman B (2011) Temporal visualization of social network dynamics: prototypes for nation of neighbors. In: Salerno J, Yang SJ, Nau D, Chai S-K (eds) Social computing, behavioral-cultural modeling and prediction. Springer, New York, pp 309–316
- Ahn JW, Plaisant C, Shneiderman B (2014) A task taxonomy for network evolution analysis. *IEEE Trans Vis Comput Graph* 20(3):365–376
- Asur S, Parthasarathy S, Ucar D (2009) An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Trans Knowl Discov Data (TKDD)* 3(4):16
- Bahmani B, Chowdhury A, Goel A (2010) Fast incremental and personalized pagerank. In: Proceedings of the international conference on very large data bases (VLDB)
- Beck F, Burch M, Diehl S, Weiskopf D (2014) The state of the art in visualizing dynamic graphs. In: EuroVis STAR 2
- Han W, Miao Y, Li K, Wu M, Yang F, Zhou L, Prabhakaran V, Chen W, Chen E (2014) Chronos: a graph engine for temporal graph analysis. In: Proceedings of the ninth European conference on computer systems. ACM, p 1
- Khurana U (2015) Historical graph data management. PhD thesis, University of Maryland
- Khurana U, Deshpande A (2013a) Efficient snapshot retrieval over historical graph data. In: Proceedings of IEEE international conference on data engineering, pp 997–1008
- Khurana U, Deshpande A (2013b) Hinge: enabling temporal network analytics at scale. In: Proceedings of the ACM SIGMOD international conference on management of data. ACM, pp 1089–1092
- Khurana U, Deshpande A (2016) Storing and analyzing historical graph data at scale. In: Proceedings of international conference on extending database technology, pp 65–76
- Khurana U, Nguyen VA, Cheng HC, Ahn Jw, Chen X, Shneiderman B (2011) Visual analysis of temporal trends in social networks using edge color coding and metric timelines. In: Proceedings of the IEEE third international conference on social computing, pp 549–554
- Kumar R, Novak J, Tomkins A (2010) Structure and evolution of online social networks. In: Faloutsos C et al (eds) Link mining: models, algorithms, and applications. Springer, New York, pp 337–357
- Leskovec J, Kleinberg J, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. *ACM Trans Knowl Discov Data (TKDD)* 1(1):2
- Macko P, Marathe VJ, Margo DW, Seltzer MI (2015) LLAMA: efficient graph analytics using large multi-versioned arrays. In: Proceedings of IEEE international conference on data engineering, pp 363–374
- Navlakha S, Kingsford C (2011) Network archaeology: uncovering ancient networks from present-day interactions. *PLoS Comput Biol* 7(4):e1001119
- Pan RK, Saramäki J (2011) Path lengths, correlations, and centrality in temporal networks. *Phys Rev E* 84(1):016105
- Ren C, Lo E, Kao B, Zhu X, Cheng R (2011) On querying historial evolving graph sequences. In: Proceedings of the international conference on very large data bases (VLDB)
- Salzberg B, Tsotras VJ (1999) Comparison of access methods for time-evolving data. *ACM Comput Surv (CSUR)* 31(2):158–221
- Tang L, Liu H, Zhang J, Nazeri Z (2008) Community evolution in dynamic multi-mode networks. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 677–685

- Tantipathananand C, Berger-Wolf T, Kempe D (2007) A framework for community identification in dynamic social networks. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, pp 717–726
- Toyoda M, Kitsuregawa M (2005) A system for visualizing and analyzing the evolution of the web with a time series of graphs. In: Proceedings of the sixteenth ACM conference on hypertext and hypermedia, pp 151–160
- Xu KS, Klinger M, Hero III AO (2011) Visualizing the temporal evolution of dynamic networks. *Stress* (X) 1:2
- Yi JS, Elmqvist N, Lee S (2010) TimeMatrix: analyzing temporal social networks using interactive matrix-based visualizations. *Int J Hum Comput Interact* 26(11–12):1031–1051

Hive

Alan F. Gates
 Hortonworks, Santa Clara, CA, USA

Definitions

Apache Hive is data warehousing software that facilitates reading, writing, and managing large data sets residing in distributed storage using SQL (Apache Hive PMC 2017).

Overview

Hive enables data warehousing in the Apache Hadoop ecosystem. It can run in traditional Hadoop clusters or in cloud environments. It can work with data sets as large as multiple petabytes. Initially Hive was used mainly for ETL and batch processing. While still supporting these use cases, it has evolved to also support data warehousing use cases such as reporting, interactive queries, and business intelligence. This evolution has been accomplished by adopting many common data warehousing techniques while adapting those techniques to the Hadoop ecosystem. It is implemented in Java.

Architecture

Hive's architecture is shown in figure 1. Not all of the components in the diagram are required in every installation. LLAP and HiveServer2 are optional; the Metastore can be run embedded in HiveServer2 or in the client's code.

Terms

HDFS: *Hadoop Distributed File System.* HDFS provides adaptors to many distributed file systems and object stores, including cloud object stores (e.g., Amazon's S3), and on-premise distributed systems (e.g., EMC's Isilon). Throughout this article where HDFS is referred to, it means use of the HDFS API. The underlying storage may be HDFS or another system.

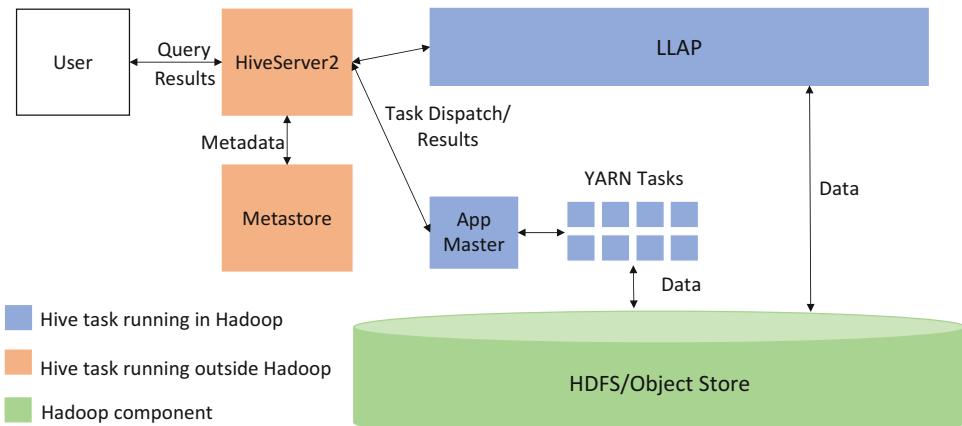
Schema: Hive uses the term schema to refer to the column names and types of data in a relation, also known as table schema. In an RDBMS context, schema often refers to a collection of relations. Throughout this article schema refers to table schema unless otherwise specified.

YARN: *Yet Another Resource Negotiator.* Hadoop's resource allocation and task manager (Vavilapalli et al. 2013). YARN is responsible for finding resources for a job in the Hadoop cluster and interacting with the job to allocate tasks and assuring the tasks and job complete and return to the submitter.

Running on Hadoop, Scalability, and High Availability

Running on Hadoop gives Hive the ability to run over large data sets with a high degree of parallelism. Hadoop clusters vary in size from a few nodes to tens of thousands of nodes. And since Hadoop provides connectors to other distributed storage systems, Hive can be used with these distributed storage systems as well. This allows Hive, on a large cluster or in the cloud, to operate on very large data sets.

Hadoop provides high availability and retry in the face of system failure for the tasks it executes.



Hive, Fig. 1 Hive components

Hive has built HiveServer2 and the Metastore to be highly available, since they do not run as tasks in Hadoop. Multiple instances of these components can be run simultaneously so that any single component failure does not render the system unavailable. Since the Metastore relies on an RDBMS to store the metadata (see below), users must choose a highly available RDBMS in order to avoid a single point of failure.

Columnar Data Storage

For traditional data warehousing queries, it is well known that columnar formats provide superior performance. There are two major columnar formats in the big data ecosystem, Apache ORC (Huai et al. 2014) and Apache Parquet. Hive supports both. ORC was originally a part of Hive and is tightly integrated into a number of Hive's newer features, such as ACID and LLAP. The rest of this section will focus on ORC, but much of what is said here applies to Parquet as well.

Columnar formats often store each column in a separate file. However, HDFS does not support co-location of related files; hence this does not work in Hadoop. Instead, ORC stores all of the columns in a single file. An ORC file is split into stripes. All of the values for a given column are stored contiguously within a stripe. These columns are compressed in type-specific ways (e.g., strings are dictionary encoded), and run length encoding is used for most types. The file

can be further compressed using standard techniques such as zlib or Snappy. Information on column offsets is placed at the end of the file so that a reader can easily read only the desired columns.

ORC also records basic statistics (e.g., minimum and maximum values) per stripe and row group (a collection of 10,000 rows in a stripe) which it uses to support predicate pushdown, allowing it to further trim the data it reads. For example, if a query requests users living in Canada and, when reading a particular row group, ORC discovers that the minimum value in the country column for that row group is Denmark, it will skip reading those rows.

“Unstructured” and Text Data

Hadoop is useful for storing very large quantities of “unstructured” data. Unstructured is a misnomer, in that this data does have structure. It is unstructured in that it has not been normalized and cleansed for storing in an RDBMS. This unstructured data may define its own structure (e.g., JSON, Apache Avro) or be a format with a predefined structure (e.g., MPEG3). It may also be a text file of comma separated values (CSV). Hive can read and write data in JSON, CSV, and Avro. It can also read pre-existing data if a user creates a table with the existing data as its data file(s).

Partitions

Given the large size of many tables, partitioning of tables is an important concept in Hive. At creation time the user specifies whether the table should be partitioned and on what key(s). All partitioning in Hive is based on value, which each partition containing a single value of the partitioning key(s). It is common, though not required, to use a date or time stamp for partitioning, as often data in Hadoop arrives at regular intervals. Each partition is stored in a separate directory in HDFS.

Most data in Hive is stored in files in HDFS. For a given table or partition, there will usually be multiple files, as each writer in a job creates its own file.

Metadata

Hive stores its data catalog in a relational database; it can use Oracle, SQL Server, MySQL, or Postgres. This service is provided by the Metastore module and is exposed via an Apache Thrift API so that other systems and tools can access the metadata. A relational database was chosen rather than storing the data in Hive itself because Hive is not an OLTP system and is not suited for the frequent small reads and writes that the Metastore executes. Fetching all of the metadata and statistics necessary to execute a query can take several seconds when the query spans thousands of partitions. To address this the Hive team is working on effective caching of the metadata so that where possible metadata and statistics can be kept in memory rather than requiring multiple database reads for every Hive query.

Query Optimization

Hive embeds and extends Apache Calcite in its optimizer. Calcite provides several features:

- A cost-based planner which uses dynamic programming as in Volcano-style optimizers. This planner is not exhaustive in its search but rather prunes the search tree to avoid an explosion of plan options.
- An exhaustive planner which explores all options and continues optimizing until no rule results in a transformation.
- A multi-phased approach to planning, where the system groups related rules into a phase. This speeds the optimization process as each phase has fewer alternatives to consider and allows the system to select the cost-based planner for some phases and exhaustive for others.
- Common optimization rules.
- The ability for systems to extend the rule set with their own rules.

Hive makes use of all these features. In particular it has added a large number of its own rules to the planner. Statistics for planning can be gathered automatically as part of data being inserted into Hive, or later by users running ANALYZE over a table or partition.

Hive is currently working on enhancing its optimizer in order to efficiently support cubing-style queries. It will do this by storing aggregate indexes in Druid (a separate open-source project that can very quickly answer many cubing-style queries) and using the optimizer to rewrite queries from the table to the associated index. Work is also underway to support materialized views so that common aggregates can be pre-materialized. Calcite includes an algorithm to help determine the appropriate aggregates to materialize.

Partition Pruning

One of the most important performance optimizations for any distributed data processing system is determining what data to read. Users are encouraged to partition their data such that most of their queries will only require some partitions. This must be balanced against creating too many small

partitions, which puts pressure on HDFS (since it has an upper bound on the number of files it can support in the file system) and the Metastore since it has to store metadata and statistics for each partition.

When a query's WHERE clause includes a predicate on the partition column(s) that can be statically computed, the Metastore can determine which partitions are required for the query. However, in the case where input data is trimmed via a join condition or subquery, Hive cannot statically determine the appropriate partitions to read. This scenario is common with fact and dimension tables. In order to efficiently execute such a query, Hive prunes partitions at runtime. When assigning segments of data to individual tasks, if it determines that the join will drop all records in the partition (based on the partition key or ORC statistics), it will remove that task from the job. This is a crucial performance optimization that allows Hive to efficiently execute data warehousing-style queries.

Query Execution

Hive offers different options for query execution. Originally all queries were executed using Hadoop's MapReduce framework. MapReduce scales very well for large data sets, but it suffers from a number of issues as an execution framework for SQL queries. Queries must be constructed as a chain of MapReduce jobs, each of which writes their results to HDFS in between. MapReduce is a very rigid model. Each job takes one input, applies the map function to it, shuffles the data between machines in order to group the records by key, applies the reduce function, and produces one output. The relational operator set can be pushed into this, but it is neither simple nor efficient.

MapReduce worked well as an engine in Hive's early days when it was focused solely on ETL and large batch jobs. As it began to transition to more data warehousing use cases, Hive needed to support more complex SQL and provide much better latency for query performance.

To address this, Hive was extended to use other Hadoop-based engines, the first of which was Apache Tez. Tez jobs are expressed as a directed acyclic graph (DAG), where nodes are processing tasks and edges are data flows between tasks. Multiple in- and outbound edges per node are supported. Tez also supports modifying the DAG at runtime based on feedback during processing (Saha et al. 2015). These features provide Hive a much simpler and more efficient model for executing its operators.

Another engine chosen was Apache Spark. Spark also supports arbitrary graphs of tasks, with intermediate data being stored in memory of the task machines. This makes it both more flexible and much faster than MapReduce (Zaharia 2010).

Hive currently supports Tez, Spark, and MapReduce, though MapReduce has been deprecated in recent versions.

The use of Tez or Spark addresses many of the performance issues seen with MapReduce, in some cases improving performance by a factor of 100 or more (Shanklin 2014). However, even with these new engines, performance issues remained due to the fact that Hadoop runs each task in a dedicated JVM. This slows down queries in three ways:

- When a job is submitted, it takes a minimum of 2–5 s to spawn the task JVMs.
- Java's just-in-time compiler (JIT) is not invoked until each task is part way through processing its data, even though the task is running the same code as many other tasks before it.
- There are limited options for caching data; each task must read its data from HDFS.

To address these issues, the Hive team developed LLAP (Live Long and Process). LLAP makes use of traditional MPP techniques. A set of long-running YARN tasks is allocated in the Hadoop cluster. Each node has a cache of data. This cache is column and ORC row group aware so that only hot portions of the data are cached.

Any node can cache any segment of the data. This prevents bottlenecks when a portion of the data is very hot and many readers wish to access it. When a job has a task ready for execution, the task is sent to an LLAP node. If the data to be accessed by that task is already cached on an LLAP node and that node has processing capacity, the task will be placed there. If the node does not have capacity, then the task will be placed on a node with capacity, which will then also cache the required data. LLAP nodes can exchange data in the same way that Tez tasks can, thus enabling the whole query to be executed in these standing servers. An area of active development for LLAP is effective workload management, as provided by many mature data warehousing systems.

MPP-like systems such as LLAP perform very well for traditional reporting and BI queries. However, they are not optimal for handling queries that need to read very large amounts of data. Consider a fact to fact join over 1 year of data where each input is a terabyte. The intermediate results will likely be quite large. It is unlikely that a full year of data will be in the cache, since most queries are over more recent data. And reading the full year into the cache will force out more valuable recent data. LLAP will not be able to dynamically add nodes to handle the large intermediate results. In contrast, this is exactly the situation where Hadoop performs well, since it can spawn sufficient tasks to handle reading and shuffling the data in parallel. In order to support interactive queries as well as large batch jobs, Hive allows users to run queries either in LLAP or use Tez, Spark, or MapReduce.

Whether running another engine such as Tez or running in LLAP, Hive provides all of the relational operators to execute its queries. These include standard operators such as projection, filter, group, sort, and join. Hive supports several types of joins:

- Broadcast join. A table, usually small enough to fit into memory, is broadcast to each task and joined against a fragment of the larger table.
- Sort-merge-bucket (SMB) join. This depends on the data already being partitioned and

sorted on the join key. In this case it will materialize the join via a merge join on each segment of data.

- Distributed hash join. This redistributes records from all inputs to a number of tasks based on the join key and then materializes the join in each task. This is by far the most expensive join type since it requires redistributing all the input data across the network. However, it scales much better than a broadcast join and does not require the data to be already sorted to be efficient like SMB join does.

Initially Hive's operators used the textbook database method of building a small set of operators that handle all data types and pull one record at a time through the operator tree. This model does not perform well for data warehousing queries. It does not efficiently make use of pipelining in the CPU, as the need to support multiple data types creates many branches in the code. It also tends to produce CPU cache misses since there is no guarantee that records are co-located in memory, forcing expensive loads from main memory. This can be addressed by operating on records in batches that are sized to fit the CPU's cache and reworking operators to be data-type specific and to minimize branches (Boncz et al. 2005). Hive reworked its operators to conform to this pattern. Hive refers to these new operators as “vectorized.”

SQL

The Hive project has stated a goal of supporting all the segments of ANSI SQL 2011 that are relevant for data warehousing operations. It has not yet achieved that goal but is tracking progress toward it (Apache Hive SQL Conformance 2017).

In addition to supporting standard SQL data types, Hive also supports additional Java-like types. These include strings (unbounded character strings, similar to RDBMS text type, but stored in row), lists (arrays of a given type), structs (a predefined set of names and types; ANSI row types), and maps (a free form mapping

of string names to values). These additional types are frequently used in the big data world as not all data is normalized, especially when first ingested.

Data Ingestion and ACID

Hive provides a LOAD statement to allow users to load external data into the system. It also supports “external tables,” where the data is written and owned by a user external to Hive. At table creation time, the user specifies where the data is located.

Hive can also write data itself via INSERT or CREATE TABLE AS SELECT statements. Originally Hive only supported writing an entire partition or table, or appending rows to an existing partition or table, though with limited ACID guarantees. Recent versions of Hive support standard transactional ACID semantics and UPDATE, DELETE, and MERGE statements. Currently all transactions are limited to a single statement; work on allowing multi-statement transactions is ongoing.

As HDFS provides write-once semantics, supporting ACID requires Hive to store changed records in side files, referred to as delta files, along with information on which transaction wrote the changed record. Transaction state is tracked in the Metastore. Readers are provided with a list of open and aborted transaction as well as their own transaction ID. From this they are able to determine which version of the record they should read. When a table’s or partition’s delta files become too numerous or too large, a compaction process automatically runs in the background to integrate the changed records into the base data files. This compaction process does not prevent simultaneous read and write of the data by users.

Hive can also ingest data from an input stream such as Apache Kafka and store it in a table with transactional semantics so that readers can see very recent data. Currently this only supports insert of new records. Adding support for update and delete of records is an area of future development and is very desirable as it will allow Hive to support change data capture (CDC).

User-Defined Functions

Given its roots in the Hadoop world, Hive has always supported user-defined functions (UDFs). These UDFs are written in Java. They allow the user to provide column transformation functions, aggregate functions (UDAFs), and table operations (UDTFs) that control how data in complex types can be expanded out to one or more rows (referred to as exploding the data).

Security

H

Authentication in Hadoop and related systems is handled by Kerberos. Using components such as Apache Knox, this authentication can be integrated with LDAP or Active Directory. Hive makes use of these features.

Authorization to read and write data in Hadoop is provided by HDFS, using a standard POSIX model. This presents challenges for Hive since it is not useful to secure the data at a SQL level if the user can access the data at the file level. There are several ways to deal with this.

Hive supports deferring all security decisions to HDFS. It refers to this as storage-based authorization. Table read operations (including metadata operations such as describing the table’s schema) are only allowed if the user has read permissions on the underlying data. Similarly, table write operations (including metadata operations such as altering the table) are only allowed if the user has write permissions on the underlying data. In this scenario Hive can only allow access to all of a table or none (i.e., there are no column or row level permissions).

As a second option, Hive supports standard SQL security with roles, grants, and revokes. For this to work properly, the user that the Hive processes are running as must own all of the data stored in HDFS.

Finally, Hive can integrate with Apache Ranger or Apache Sentry. These provide authorization across the Hadoop ecosystem, assuring that a user’s permissions in Hive are coordinated with his or her permissions in HDFS

and other components. Via plug-ins to Hive, Ranger or Sentry can examine the query plan and authorize or reject a query. They can also choose to modify the query plan. For example, Ranger uses this feature to anonymize sensitive data before returning it to the user.

Since Hive runs in Hadoop, it also has to decide what user to execute its jobs as. Traditionally Hadoop jobs run as the user submitting the job. When running with Tez, Spark, or MapReduce, Hive can follow this paradigm, which it refers to as running in “doAs” mode, since it is doing the operations as the user. This works well in environments where a number of different tools are being used to read and write data. It also allows the use of UDFs. However, security options are limited in this scenario, since Hive is dependent on the ability of the user to access the files that store the data.

Hive can also run all jobs as a dedicated user, referred to as the “hive user.” This gives the ability to provide fine-grained security, since Hive owns all of the data and thus controls access to it. However, it is not safe to allow UDFs when running as a super user. Only system-provided functions and administrator “blessed” UDFs are allowed in this scenario.

Client Connections

Hive includes a JDBC driver that works with HiveServer2. The Hive project does not provide an ODBC driver, but there are commercial ones available.

Hive includes a command line tool called beeline which provides an interactive SQL shell. Beeline connects to HiveServer2 via JDBC.

Hive also supports a command line tool that allows for interactive SQL operations. It is not recommended for production environments since it requires direct access to the Hadoop cluster and HDFS files.

References

Apache Hive (2017) <http://hive.apache.org/>

- Apache Hive SQL Conformance (2017) <https://cwiki.apache.org/confluence/display/Hive/Apache+Hive+SQL+Conformance>. Accessed 11 Nov 2017
- Boncz P et al (2005) MonetDB/X100: hyper-pipelining query execution. In: Proceedings of the 2005 CIDR conference, Asilomar, pp 225–237
- Huai Y et al (2014) Major technical advancements in Apache Hive. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, Snowbird Utah, 22–27 June 2014
- Saha B et al (2015) Apache Tez: a unifying framework for modeling and building data processing applications. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, Melbourne, 31 May–4 June 2015
- Shanklin C (2014) Benchmarking Apache Hive 13 for enterprise Hadoop. <https://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop/>. Accessed 9 Nov 2017
- Vavilapalli V et al (2013) Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing, Santa Clara, 1–3 Oct 2013
- Zaharia M (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing, Boston, 22–25 June 2010

Holistic Schema Matching

Erhard Rahm and Eric Peukert
University of Leipzig, Leipzig, Germany

Synonyms

[Collective schema matching](#)

Definitions

Holistic schema matching aims at identifying semantically corresponding elements in multiple schemas, e.g., database schemas, web forms, or ontologies. The corresponding elements from N (>2) sources are typically grouped into clusters with up to N members. Holistic schema matching is usually applied when multiple schemas need to be combined within an integrated schema or ontology.

Overview

Holistic schema matching aims at identifying semantically corresponding elements in multiple (>2) schemas, such as database schemas, web forms, or ontologies. It is to be contrasted with the traditional pairwise schema matching (Rahm and Bernstein 2001; Euzenat and Shvaiko 2013) between two input schemas only that determines a so-called mapping consisting of a set of correspondences, i.e., pairs of elements of the input schemas (table attributes, ontology concepts) that match with each other. Holistic schema matching is applied to more than two input schemas so that matching schema elements are clustered or grouped together.

This form of schema matching is needed to support data integration for more than two data sources and is thus of increasing relevance for big data applications. There is a large spectrum of possible use forms for holistic schema matching depending on the number and complexity of schemas. Traditional approaches for schema and ontology integration (Batini et al. 1986; Raunich and Rahm 2014) are typically limited to few sources with a high degree of manual interaction. The need for automatic approaches has increased by the availability of many web data sources and ontologies. In domains such as the life sciences, there are numerous overlapping ontologies so that there is an increasing need to semantically integrate these ontologies into a combined ontology to consistently represent the knowledge of a domain. An example of such an integration effort is the biomedical ontology UMLS Metathesaurus (Bodenreider 2004) which currently combines more than 3 million concepts and more than 12 million synonyms from more than 100 biomedical ontologies and vocabularies. The integration process is highly complex, and still largely dependent on domain experts, in order to identify the synonymous concepts across all source ontologies as well as to derive a consistent ontology structure for these concepts and their relations. Related to the integration of multiple ontologies is the construction of so-called *knowledge graphs* from different web sources and ontologies (Dong et al. 2014) that requires to holistically integrate

both entities of one or several domains (e.g., countries, cities, presidents, movies, etc.) and schema elements such as attributes, relationships, and concepts.

While the challenges for integrating very large sets of schemas are far from being solved, there are initial approaches for holistic schema matching that we will discuss in this entry, in particular for the following use cases (Rahm 2016):

- Integration of several database schemas or ontologies
- Matching and integrating multiple *web forms*, e.g., for a mediated web search over several databases
- Matching several *web tables* of a certain domain e.g., for improved query processing

Ideally, holistic schema matching is largely automatic with minimal user interaction and achieves a high match quality, i.e., it correctly clusters matching schema elements. Furthermore, holistic match approaches need to be efficient and scalable to many schemas/sources. It should also be easily possible to add and utilize additional schemas and deal with changes in the source schemas.

As we will see, holistic schema matching still uses pairwise matching as a building block before a clustering takes place. However, scalability to many sources is not feasible if a pairwise mapping of all N input schemas is needed since this would result in a quadratic number of mappings, e.g., almost 20,000 mappings for 200 sources (Rahm 2016).

Key Research Findings

In the following, we will discuss main approaches for the mentioned use cases.

Holistic Matching for Schema and Ontology Integration

To match and integrate multiple schemas or ontologies, one can in principle apply a pairwise matching and integration (or merging) multiple

times in an incremental way. For instance, one can use one of the schemas as the initial integrated schema and incrementally match and merge the next source with the intermediate result until all source schemas are integrated. This has the advantage that only $N-1$ match and integration steps are needed for N schemas.

Such binary integration strategies have already been considered in early approaches to schema integration (Batini et al. 1986). More recently it has been applied within the Porsche approach (Saleem et al. 2008) to automatically merge many tree-structured XML schemas. The approach holistically clusters all matching elements in the nodes of the integrated schema. The placement of new source elements not found in the (intermediate) integrated schema is based on a simplistic heuristic only. A general problem of incremental merge approaches is that the final merge result typically depends on the order in which the input schemas are matched and merged.

A full pairwise matching has been applied in (Hu et al. 2011) to match the terms of more than 4000 web-extracted ontologies. The match process using a state-of-the-art match tool took about 1 year on six computers showing the insufficient scalability of unrestricted pairwise matching.

A holistic matching of concepts in linked sources of the so-called Data Web has been proposed in (Gruetze et al. 2012). The authors first cluster the concepts (based on keywords from their labels and descriptions) within different topical groups and then apply pairwise matching of concepts within groups to finally determine clusters of matching concepts. The approach is a good first step to automatic holistic schema matching but suffered from coverage and scalability limitations. So topical grouping was possible for less than 17% of the considered one million concepts, and matching for the largest group of 5 K concepts took more than 30 h.

The matching between many schemas/ontologies can be facilitated by the reuse of previously determined mappings (links) between such sources, especially if such mappings are available in the Data Web or in repositories such as Bio-Portal and LinkLion (Nentwig et al. 2014).

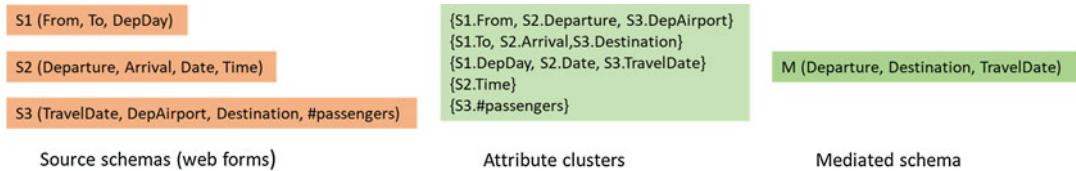
Such a reuse of mappings has already been proposed for pairwise schema and ontology matching based on a repository of schemas and mappings (Do and Rahm 2002; Madhavan et al. 2005; Rahm 2011). A simple and effective approach is based on the composition of existing mappings to quickly derive new mappings. In particular, one can derive a new mapping between schemas S_1 and S_2 by composing existing mappings, e.g., mappings between S_1 and S_i and between S_i and S_2 for any intermediate schema S_i . Such composition approaches have been investigated in (Gross et al. 2011) and were shown to be very fast and also effective. A promising strategy is to utilize a hub schema (ontology) per domain, such as UMLS in the biomedical domain, to which all other schemas are mapped. Then one can derive a mapping between any two schemas by composing their mappings with the hub schema.

Holistic Matching of Web Forms and Web Tables

The holistic integration of many schemas has mainly been studied for simple schemas such as web forms and web tables typically consisting of only a few attributes. Previous work for web forms concentrated on their integration within a mediated schema, while for web tables, the focus has been on the semantic annotation and matching of attributes.

The integration of *web forms* has been studied to support a meta-search across deep web sources, e.g., for comparing products from different online shops. Schema integration mainly entails grouping or clustering similar attributes from the web forms, which is simpler than matching and merging complex schemas. As a result, scalability to dozens of sources is typically feasible. Figure 1 illustrates the main idea for three simple web forms to book flights. The attributes in the forms are first clustered, and then the mediated schema is determined from the bigger clusters referring to attributes available in most sources. Queries on the mediated schema can then be answered on most sources.

Proposed approaches for the integration of web forms include WISE-Integrator and MetaQuerier (He et al. 2004; He and Chang



Holistic Schema Matching, Fig. 1 Integration of web forms (query interfaces) into a mediated schema for flight bookings

2003). Matching and clustering of attributes is mainly based on the linguistic similarity of the attribute names (labels). The approaches also observe that similarly named attributes co-occurring in the same schema (e.g., FirstName and LastName) do not match and should not be clustered together. Das Sarma et al. (2008) proposes the automatic generation of a so-called probabilistic mediated schema from N input schemas, which is in effect a ranked list of several mediated schemas.

A much larger number (thousands and millions) of sources have to be dealt with in the case of web-extracted datasets like *web tables* made available within open data repositories such as data.gov or datahub.io. The collected datasets are typically from diverse domains and initially not integrated at all. To enable their usability, e.g., for query processing or web search, it is useful to group the datasets into different domains and to semantically annotate attributes.

Several approaches have been proposed for such a semantic annotation and enrichment of attributes by linking them to concepts of knowledge graphs like YAGO, DBpedia, or Probbase (Limaye et al. 2010; Wang et al. 2012). In (Balakrishnan et al. 2015), attributes are mapped to concepts of the Google Knowledge Graph (which is based on Freebase) by mapping the attribute values of web tables to entities in the knowledge graph and thus to the concepts of these entities.

The enriched attribute information as well as the attribute instances can be used to match and cluster related web tables to facilitate the combined use of their information. This has only been studied to a limited degree so far. In particular, the Infogather system (Yakout et al. 2012) utilizes enriched attribute information to match web tables with each other. To limit the scope, they

determine topic-specific schema match graphs that only consider schemas similar to a specific query table. The match graphs help to determine matching tables upfront before query answering and to holistically utilize information from matching tables. Eberius et al. (2013) uses instance-based approaches to match the attributes of web tables considering the degree of overlap in the attribute values, but they do utilize the similarity of attribute names or their annotations.

The holistic integration of both web forms and web tables is generally only relevant for schemas of the same application domain. For a very large number of such schemas, it is thus important to first categorize schemas by domain. Several approaches have been proposed for the automatic domain categorization problem of web forms (Barbosa 2007; Mahmoud and Aboulnaga 2010) typically based on a clustering of attribute names and the use of further features such as explaining text in the web page where the form is placed. While some approaches (Barbosa 2007) considered the domain categorization for only few predefined domains, Mahmoud and Aboulnaga (2010) cluster schemas into a previously unknown number of domain-like groups that may overlap. This approach has also been used in Eberius et al. (2013) for a semantic grouping of related web tables. The categorization of web tables should be based on semantically enriched attribute information, the attribute instances, and possibly information from the table context in the web pages (Balakrishnan et al. 2015). The latter study observed that a large portion of web tables includes a so-called “subject” attribute (often the first attribute) indicating the kinds of entities represented in a table (companies, cities, etc.) which allows already a rough categorization of the tables.

Directions for Future Research

Compared to the huge amount of previous work on pairwise schema matching, research on holistic schema matching is still at an early stage. As a result, more research is needed in improving the current approaches for a largely automatic, effective, and efficient integration of many schemas and ontologies. In addition to approaches for determining an initial integrated schema/ontology, there is a need for incremental schema integration approaches that can add new schemas without having to completely reintegrate all source schemas. The use of very large sets of web-extracted datasets such as web tables also needs more research to categorize and semantically integrate these datasets.

To deal with a very large number of input schemas, it is important to avoid the quadratic complexity of a pairwise mapping between all input schemas. One approach, which is especially suitable for integrating complex schemas and ontologies, is to incrementally match and integrate the input schemas. More research is here needed to evaluate how known approaches for pairwise matching and integration could be used or extended for such a setting.

For simpler schemas, or if only a clustering of concepts/attributes is aimed at, grouping of the input schemas by domain or mutual similarity is able to reduce the overall complexity since matching can then be restricted to the schemas of the same group. To limit the effort for matching schema elements within such groups, performance techniques from entity resolution such as blocking (Koepcke and Rahm 2010) can further be utilized. More research is thus needed to investigate these ideas for a very large number of schemas/datasets.

Cross-References

- ▶ [Large-Scale Entity Resolution](#)
- ▶ [Large-Scale Schema Matching](#)

References

- Balakrishnan S, Halevy AY, Harb B, Lee H, Madhavan J, Rostamizadeh A, Shen W, Wilder K, Wu F, Yu C (2015) Applying web tables in practice. In: Proceedings of the CIDR
- Barbosa L, Freire J, Silva A (2007) Organizing hidden-web databases by clustering visible web documents. In: Proceedings of the international conference on data engineering, pp 326–335
- Batini C, Lenzerini M, Navathe SB (1986) A comparative analysis of methodologies for database schema integration. ACM Comput Surv 18(4):323–364
- Bodenreider O (2004) The unified medical language system (UMLS): integrating bio-medical terminology. Nucleic Acids Res 32(suppl 1):D267–D270
- Das Sarma A, Dong X, Halevy AY (2008) Bootstrapping pay-as-you-go data integration systems. In: Proceedings of the ACM SIGMOD conference
- Do HH, Rahm E (2002) COMA – a system for flexible combination of schema matching approaches. In: Proceedings of the international conference on very large data bases, pp 610–621
- Dong X, Gabrilovich E, Heitz G, Horn W, Lao N, Murphy K, Strohmann T, Sun S, Zhang W (2014) Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of KDD, New York, USA, pp 601–610
- Eberius J, Damme P, Braunschweig K, Thiele M, Lehner W (2013) Publish-time data integration for open data platforms. In: Proceedings of the ACM workshop on open data
- Euzenat J, Shvaiko P (2013) Ontology matching, 2nd edn. Springer, Berlin/Heidelberg, Germany
- Gross A, Hartung M, Kirsten T Rahm E (2011) Mapping composition for matching large life science ontologies. In: Proceedings of the ICBO, pp 109–116
- Gruetze T, Boehm C, Naumann F (2012) Holistic and scalable ontology alignment for linked open data. In: Proceedings of linked data on the web
- He B, Chang KC (2003) Statistical schema matching across web query interfaces. In: Proceedings of the ACM SIGMOD conference, pp 217–228
- He H, Meng W, Yu CT, Wu Z (2004) Automatic integration of web search interfaces with WISE-integrator. VLDB J 13(3):256–273
- Hu W, Chen J, Zhang H, Qu Y (2011) How matchable are four thousand ontologies on the semantic web. In: Proceedings of the ESWC, Springer LNCS 6643
- Koepcke H, Rahm E (2010) Frameworks for entity matching: a comparison. Data Knowl Eng 69(2):197–210
- Limaye G, Sarawagi S, Chakrabarti S (2010) Annotating and searching web tables using entities, types and relationships. PVLDB 3(1–2):1338–1347
- Madhavan J, Bernstein PA, Doan A, Halevy AY (2005) Corpus-based schema matching. In: Proceedings of the IEEE ICDE conference
- Mahmoud HA, Aboulnaga A (2010) Schema clustering and retrieval for multi-domain pay-as-you-go data

- integration systems. In: Proceedings of the ACM SIGMOD conference
- Nentwig M, Soru T, Ngonga Ngomo A, Rahm E (2014) LinkLion: a link repository for the web of data. In: Proceedings of the ESWC (Satellite Events), Springer LNCS 8798
- Rahm E (2011) Towards large-scale schema and ontology matching. In: Schema matching and mapping. Springer, Berlin/Heidelberg, Germany
- Rahm E (2016) The case for holistic data integration. In: Proceedings of the ADBIS conference, Springer LNCS 9809
- Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350
- Raunich S, Rahm E (2014) Target-driven merging of taxonomies with ATOM. Inf Syst 42:1–14
- Saleem K, Bellahsene Z, Hunt E (2008) PORSCHE: performance oriented SCHEma mediation. Inf Syst 33(7–8):637–657
- Wang J, Wang H, Wang Z, Zhu KQ (2012) Understanding tables on the web. In: Proceedings of the ER conference, Springer LNCS 7532, pp 141–155
- Yakout M, Ganjam K, Chakrabarti K, Chaudhuri S (2012) Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In: Proceedings of the ACM SIGMOD conference, pp 97–108

HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases

Salman Niazi, Mahmoud Ismail, Seif Haridi, and Jim Dowling
KTH – Royal Institute of Technology,
Stockholm, Sweden

Definitions

Modern NewSQL database systems can be used to store fully normalized metadata for distributed hierarchical file systems, and provide high throughput and low operational latencies for the file system operations.

Introduction

For many years, researchers have investigated the use of database technology to manage file system metadata, with the goal of providing extensible

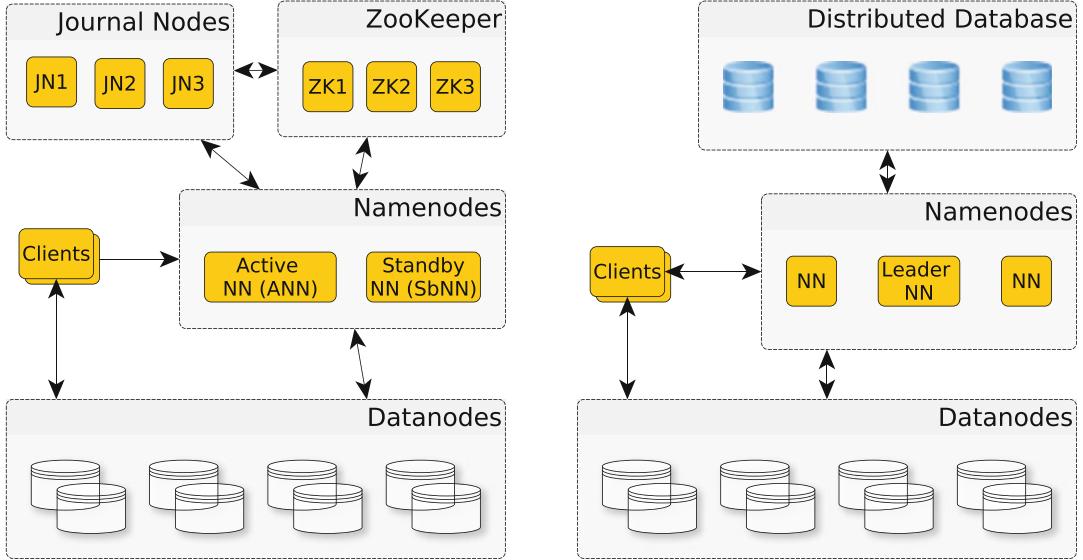
typed metadata and support for fast, rich metadata search. However, previous attempts failed mainly due to the reduced performance introduced by adding database operations to the file system's critical path. However, recent improvements in the performance of distributed in-memory online transaction processing databases (NewSQL databases) led us to reinvestigate the possibility of using a database to manage file system metadata, but this time for a distributed, hierarchical file system, the Hadoop file system (HDFS). The single-host metadata service of HDFS is a well-known bottleneck for both the size of HDFS clusters and their throughput. In this entry, we characterize the performance of different NewSQL database operations and design the metadata architecture of a new drop-in replacement for HDFS using, as far as possible, only those high-performance NewSQL database access patterns. Our approach enabled us to scale the throughput of all HDFS operations by an order of magnitude.

H

HopsFS vs. HDFS

In both systems, namenodes provide an API for metadata operations to the file system clients, see Fig. 1. In HDFS, an active namenode (ANN) handles client requests, manages the metadata in memory (on the heap of a single JVM), logs changes to the metadata to a quorum of journal servers, and coordinates repair actions in the event of failures or data corruption. A standby namenode asynchronously pull the metadata changes from the journal nodes and applies the changes to its in memory metadata. A ZooKeeper service is used to signal namenode failures, enabling both agreement on which namenode is active as well as failover from active to standby.

In HopsFS, namenodes are stateless servers that handle client requests and process the metadata that is stored in an external distributed database, NDB in our case. The internal management (housekeeping) operations must be coordinated among the namenodes. HopsFS solves this problem by electing a leader namenode that is responsible for the housekeeping.



HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases, Fig. 1 In HDFS, a single namenode manages the namespace metadata. For high availability, a log of metadata changes is stored on a set of journal nodes using quorum-based replication. The log is subsequently replicated asynchronously to a standby

We use the database as a shared memory to implement a leader election and membership management service (Salman Niazi et al. 2015).

In both HDFS and HopsFS, datanodes are connected to all the namenodes. Datanodes periodically send a heartbeat message to all the namenodes to notify them that they are still alive. The heartbeat also contains information such as the datanode capacity, its available space, and its number of active connections. Heartbeats update the list of datanode descriptors stored at namenodes. The datanode descriptor list is used by namenodes for future block allocations, and it is not persisted in either HDFS or HopsFS, as it is rebuilt on system restart using heartbeats from datanodes.

HopsFS clients support random, round-robin, and sticky policies to distribute the file system operations among the namenodes. If a file system operation fails, due to namenode failure or overloading, the HopsFS client transparently retries the operation on another namenode (after backing off, if necessary). HopsFS clients refresh the namenode list every few seconds, enabling new

namenode. In contrast, HopsFS supports multiple stateless namenodes and a single leader namenode that all access metadata stored in transactional shared memory (NDB). In HopsFS, leader election is coordinated using NDB, while ZooKeeper coordinates leader election for HDFS

namenodes to join an operational cluster. HDFS v2.X clients are fully compatible with HopsFS, although they do not distribute operations over namenodes, as they assume there is a single active namenode.

MySQL's NDB Distributed Relational Database

HopsFS uses MySQL's Network Database (NDB) to store the file system metadata. NDB is an open-source, real-time, in-memory, shared nothing, distributed relational database management system.

NDB cluster consists of three types of nodes: NDB datanodes that store the tables, management nodes, and database clients; see Fig. 3.

The management nodes are only used to disseminate the configuration information and to detect network partitions. The client nodes access the database using the SQL interface via MySQL Server or using the native APIs implemented in C++, Java, JPA, and JavaScript/Node.js.

Figure 3 shows an NDB cluster setup consisting of four NDB datanodes. Each NDB datanode consists of multiple transaction coordinator (TC) threads, which are responsible for performing two-phase commit transactions; local data management threads (LDM), which are responsible for storing and replicating the data partitions assigned to the NDB datanode; send and receive threads, which are used to exchange the data between the NDB datanodes and the clients; and IO threads which are responsible for performing disk IO operations.

MySQL Cluster horizontally partitions the tables, that is, the rows of the tables are uniformly distributed among the database partitions stored on the NDB datanodes. The NDB datanodes are organized into node replication groups of equal sizes to manage and replicate the data partitions. The size of the node replication group is the replication degree of the database. In the example setup, the NDB replication degree is set to two (default value); therefore, each node replication group contains exactly two NDB datanodes. The first replication node group consists of NDB datanodes 1 and 2, and the second replication node group consists of NDB datanodes 3 and 4. Each node group is responsible for storing and replicating all the data partitions assigned to the NDB datanodes in the replication node group.

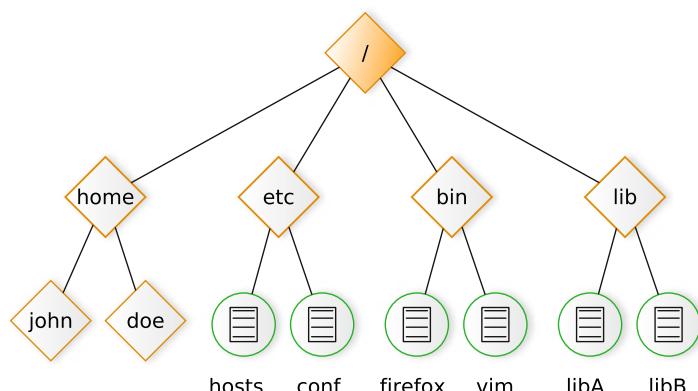
By default, NDB hashes the primary key column of the tables to distribute the table's rows among the different database partitions. Figure 3 shows how the inodes table for the namespace shown in Fig. 2 is partitioned and

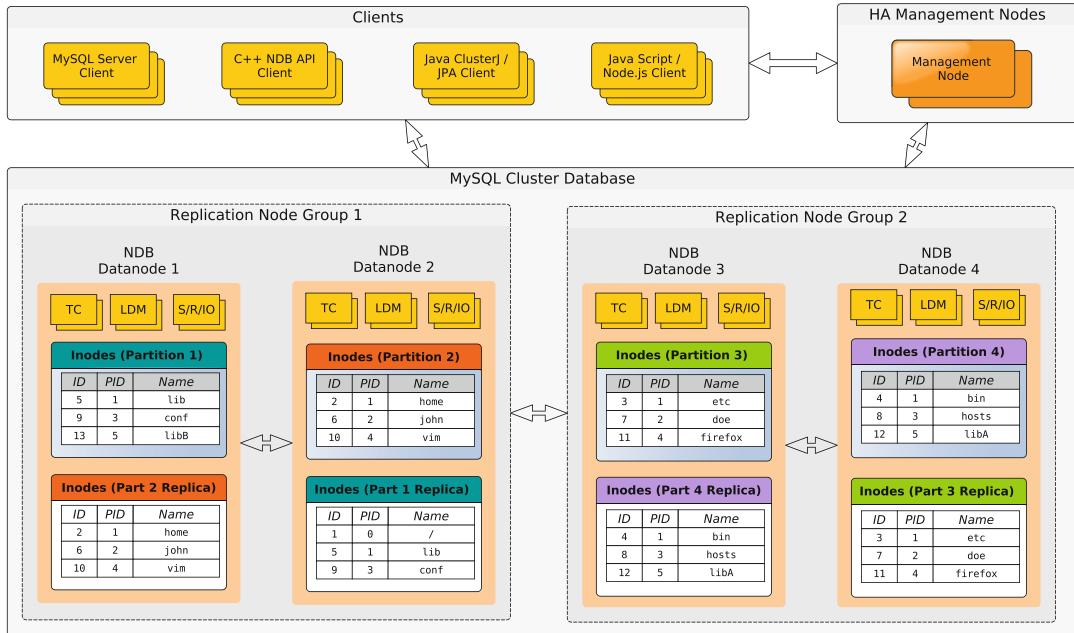
stored in the NDB cluster database. In production deployments each NDB datanode may store multiple data partitions, but, for simplicity, the datanodes shown in Fig. 3 store only one data partition. The replication node group 1 is responsible for storing and replicating partitions 1 and 2 of the inodes' table. The primary replicas of partition 1 is stored on the NDB datanode 1, and the replica of the partition 1 is stored on the other datanode of the same replication node group, that is, NDB datanode 2. NDB also supports user-defined partitioning of the stored tables, that is, it can partition the data based on a user-specified table column. User-defined partitioning provides greater control over how the data is distributed among different database partitions, which helps in implementing very efficient database operations.

NDB only supports *read-committed* transaction isolation level which is not sufficient for implementing practical applications. However, NDB supports row-level locking which can be used to isolate transactions operating on the same datasets. NDB supports *exclusive* (write) locks, *shared* (read) locks, and *read – committed* (read the last committed value) locks. Using these locking primitives, HopsFS isolates different file system operations trying to mutate the same subset of inodes.

HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases, Fig. 2 A

sample file system
namespace. The diamond
shapes represent directories
and the circles represent
files





HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases, Fig. 3 System architecture diagram of MySQL's Network Database (NDB) cluster.

The NDB cluster consists of three types of nodes, that is, database clients, NDB management nodes, and NDB database datanodes

Types of Database Operations

A transactional file system operation consists of three distinct phases. In the first phase, all the metadata that is required for the file system operation is read from the database; in the second phase, the operation is performed on the metadata; and in the last phase, all the modified metadata (if any) is stored back in the database. As the latency of the file system operation greatly depends on the time spent in reading and writing the data, therefore, it is imperative to understand the latency and computational cost of different types of database operations, used to read and update the stored data, in order to understand how HopsFS implements low-latency scalable file system operations. The data can be read using four different types of database operations, such as *primary key*, *partition-pruned index scan*, *distributed index scan*, and *distributed full table scan* operations. NDB also supports *user-defined data partitioning* and *data distribution-aware transactions* to reduce the latency of the distributed transactions. A brief description of these operations is as follows:

Application-Defined Partitioning (ADP)

NDB allows developers to override the default partitioning scheme for the tables, enabling a fine-grained control on how the tables' data is distributed across the data partitions. HopsFS leverage this feature, for example, the inodes table is partitioned by the parent inode ID, that is, all immediate children of a directory reside on the same database partition, enabling an efficient directory listing operation.

Distribution-Aware Transactions (DAT)

NDB supports *distribution-aware transactions* by specifying a *transaction hint* at the start of the transaction. This enlists the transaction with a transaction coordinator on the database node that holds the data required for the transaction, which reduces the latency of the database operation. The choice of the transaction hint is based on the *application-defined partitioning* scheme. Incorrect hints do not affect the correctness of the operation, since the transaction coordinator (TC) will route the requests to different NDB

datanode that holds the data; however, it will incur additional network traffic.

Primary Key (PK) Operation

PK operation is the most efficient operation in NDB that reads/writes/updates a single row stored in a database shard.

Batched Primary Key Operations

Batch operations use batching of PK operations to enable higher throughput while efficiently using the network bandwidth.

Partition-Pruned Index Scan (PPIS)

PPIS is a distribution-aware operation that exploits the *distribution-aware transactions* and *application-defined partitioning* features of NDB, to implement a scalable index scan operation such that the scan operation is performed on a single database partition.

Distributed Index Scan (DIS)

DIS is an index scan operation that executes on all database shards. It is not a distribution-aware operation, causing it to become increasingly slow for increasingly larger clusters.

Distributed Full Table Scan (DFTS)

DFTS operations are not distribution aware and do not use any index; thus, they read all the rows for a table stored in all database shards. *DFTS* operations incur high cpu and network costs; therefore, these operations should be avoided in implementing any database application.

Comparing Different Database Operations

Figure 4 shows the comparison of the throughput and latency of different database operations. These micro benchmarks were performed on a four-node NDB cluster setup running NDB version 7.5.6. All the experiments were run on premise using Dell PowerEdge R730xd servers (Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz, 256 GB RAM, 4 TB 7200 RPM HDD) connected using a single 10 GbE network adapter. We ran 400 concurrent database clients, which were uniformly distributed across 20 machines. The

experiments are repeated twice. The first set of experiments consist of read-only operations, and the second set of experiments consist of read-write operations where all the data that is read is updated and stored back in the database.

Distributed full table scan operations and distributed index scan operations do not scale, as these operations have the lowest throughput and highest end-to-end latency among all the database operations; therefore, these operations must be avoided in implementing file system operations. Primary key, batched primary key operations, and partition-pruned index scan operations are scalable database operations that can deliver very high throughput and have low end-to-end operational latency.

The design of the database schema, that is, how the data is laid out in different tables, the design of the primary keys of the tables, types/number of indexes for different columns of the tables, and data partitioning scheme for the tables, plays a significant role in choosing an appropriate (efficient) database operation to read/update the data, which is discussed in detail in the following sections.

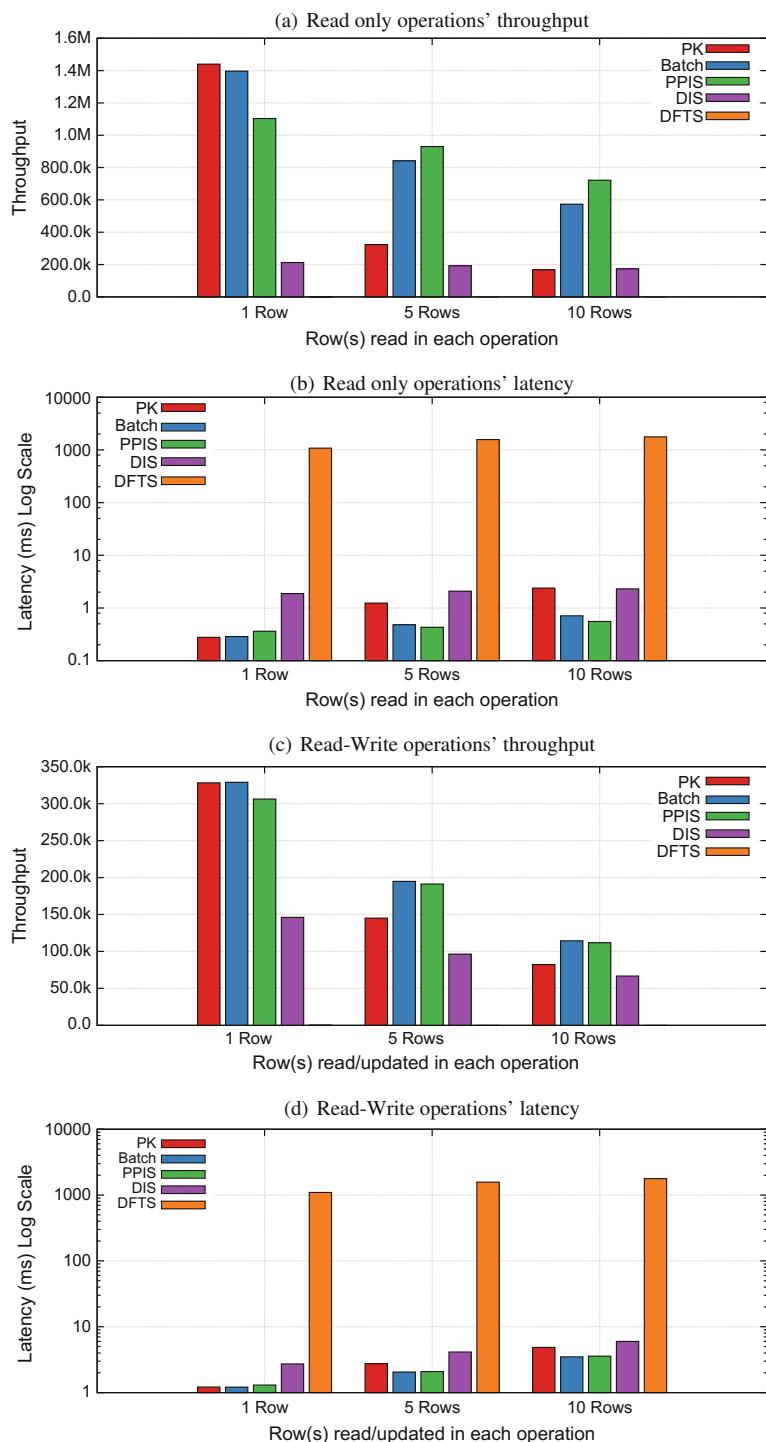
HopsFS Distributed Metadata

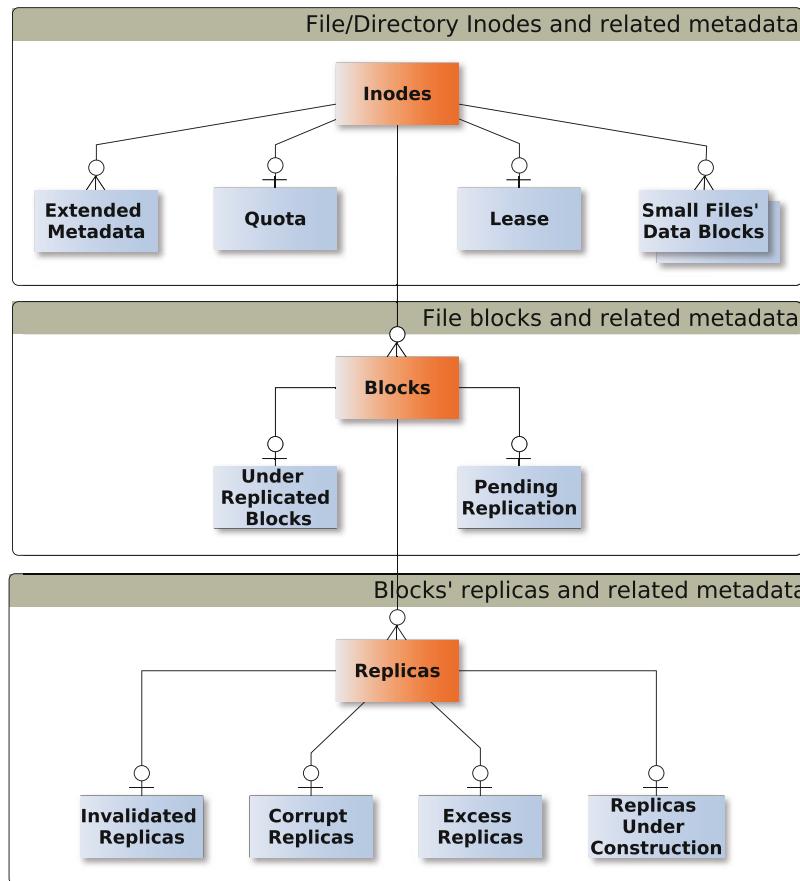
In HopsFS the metadata is stored in the database in normalized form, that is, instead of storing the full file paths with each inode as in Thomson and Abadi (2015), HopsFS stores individual file path components. Storing the normalized data in the database has many advantages, such as *rename* file system operation can be efficiently implemented by updating a single row in the database that represents the inode. If complete file paths are stored for each inode, then it would not only consume significant amount of precious database storage, but also, a simple directory rename operation would require updating the file paths for all the children of that directory subtree.

Key tables in HopsFS metadata schema are shown in the entity relational (ER) diagram in Fig. 5. The inodes for files and directories are stored in the *inodes* table. Each inode is represented by a single row in the table, which also

HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases

Fig. 4 The comparison of the throughput and latency of different database operations. (a) Read-only operations' throughput. (b) Read-only operations' latency. (c) Read-write operations' throughput. (d) Read-write operations' latency





HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases, Fig. 5 Entity relational diagram for HopsFS schema

stores information such as inode ID, name, parent inode ID, permission attributes, user ID, and size. As HopsFS does not store complete file paths with each inode, the inode's parent ID is used to traverse the inodes table to discover the complete path of the inode. The extended metadata for the inodes is stored in the *extended metadata* entity. The *quota* table stores information about disk space consumed by the directories with quota restrictions enabled. When a client wants to update a file, then it obtains a lease on the file, and the file lease information is stored in the *lease* table. Small files that are few kilobytes in size are stored in the database for low-latency access. Such files are stored in the *small files' data blocks* tables.

Non-empty files may contain multiple blocks stored in the *blocks* table. The location of each replica of the block is stored in the *replicas* table. During its life cycle a block goes through various phases. Blocks may be under-replicated if a datanode fails, and such blocks are stored in the *under-replicated blocks* table. HopsFS periodically checks the health of all the data blocks. HopsFS ensures that the number of replicas of each block does not fall below the threshold required to provide high availability of the data. The replication monitor sends commands to datanodes to create more replicas of the under-replicated blocks. The blocks undergoing replication are stored in the *pending replication blocks* table. Similarly, a

replica of a block has various states during its life cycle. When a replica gets corrupted due to disk errors, it is moved to the *corrupted replicas* table. The replication monitor will create additional copies of the corrupt replicas by copying healthy replicas of the corresponding blocks, and the corrupt replicas are moved to *invalidated replicas* table for deletion. Similarly, when a file is deleted, the replicas of the blocks belonging to the file are moved to the *invalidated replicas* table. Periodically the namenodes read the invalidated replicas tables and send commands to the datanodes to permanently remove the replicas from the datanodes' disk. Whenever a client writes to a new block's replica, this replica is moved to the *replica under construction* table. If too many replicas of a block exist (e.g., due to recovery of a datanode that contains blocks that were re-replicated), the extra copies are stored in the *excess replicas* table.

Metadata Partitioning

The choice of partitioning scheme for the hierarchical namespace is a key design decision for distributed metadata architectures. HopsFS uses user-defined partitioning for all the tables stored in NDB. We base our partitioning scheme on the expected relative frequency of HDFS operations in production deployments and the cost of different database operations that can be used to implement the file system operations. Read-only metadata operations, such as *list*, *stat*, and *file read* operations, alone account for $\approx 95\%$ of the operations in Spotify's HDFS cluster (Niazi et al. 2017). Based on the micro benchmarks of different database operations, we have designed the metadata partitioning scheme such that frequent file system operations are implemented using the scalable database operations such as primary key, batched primary key operations, and partition-pruned index scan operations. Index scans and full table scans are avoided, where possible, as they touch all database partitions and scale poorly.

HopsFS partitions *inodes* by their *parents' inode IDs*, resulting in inodes with the same parent inode being stored on the same database partitions. That enables efficient implementation

of the directory listing operation. When listing files in a directory, we use a hinting mechanism to start the transaction on a transaction coordinator located on the database partitions that holds the child inodes for that directory. We can then use a pruned index scan to retrieve the contents of the directory locally. File inode-related metadata, that is, blocks and replicas and their states, is partitioned using the file's *inode ID*. This results in metadata for a given file all being stored in a single database partition, again enabling efficient file operations. Note that the file inode-related entities also contain the inode's foreign key (not shown in the entity relational diagram Fig. 5) that is also the partition key, enabling HopsFS to read the file inode-related metadata using partition-pruned index scans.

It is important that the partitioning scheme distribute the metadata evenly across all the database partitions. Uneven distribution of the metadata across all the database partitions leads to poor performance of the database. It is not uncommon for big data applications to create tens of millions of files in a single directory (Ren et al. 2013; Patil et al. 2007). HopsFS metadata partitioning scheme does not lead to hot spots as the database partition that contains the immediate children (files and directories) of a parent directory only stores the names and access attributes, while the inode-related metadata (such as blocks, replicas, and associated states) that comprises the majority of the metadata is spread across all the database partitions because it is partitioned by the inode IDs of the files. For hot files/directories, that is, file/directories that are frequently accessed by many of concurrent clients, the performance of the file system will be limited by the performance of the database partitions that hold the required data for the hot files/directories.

HopsFS Transactional Operations

File system operations in HopsFS are divided into two main categories, that is, non-recursive file system operations that operate on a single file or directory, also called **inode operations**, and recursive file system operations that operate

on directory subtrees which may contain large number of descendants, also called **subtree operations**. The file system operations are divided into these two categories because databases, for practical reasons, put a limit on the number of read and write operations that can be performed in a single database transaction, that is, a large file system operation may take a long time to complete and the database may timeout the transaction before the operation successfully completes. For example, creating a file, mkdir, or deleting file are *inode operations* that read and update limited amount of metadata in each transactional file system operation. However, recursive file system operations such as delete, move, chmod on large directories may read and update millions of rows. As subtree operations cannot be performed in a single database operation, HopsFS breaks the large file system operation into small transactions. Using application-controlled locking for the subtree operations, HopsFS ensures that the semantics and consistency of the file system operation remain the same as in HDFS.

Inode Hint Cache

In HopsFS all file system operations are path based, that is, all file system operations operate on file/directory paths. When a namenode receives a file system operation, it traverses the file path to ensure that the file path is valid and the client is authorized to perform the given file system operation. In order to recursively resolve the file path, HopsFS uses primary key operations to read each constituent inode one after the other. HopsFS partitions the inodes' table using the parent ID column, that is, the constituent inodes of the file path may reside on different database partitions. In production deployments, such as in case of Spotify, the average file system path length is 7, that is, on average it would take 7 round trips to the database to resolve the file path, which would increase the end-to-end latency of the file system operation. However, if the primary keys of all the constituent inodes for the path are known to the namenode in advance, then all the inodes can be read from the database in a single-batched primary key operation. For reading multiple rows batched primary key operations scale better than

iterative primary key operations and have lower end-to-end latency. Recent studies have shown that in production deployments, such as in Yahoo, the file system access patterns follow a heavy-tailed distribution, that is, only 3% of the files account for 80% of file system operations (Abad 2014); therefore, it is feasible for the namenodes to cache the primary keys of the recently accessed inodes. The namenodes store the primary keys of the inodes table in a local least recently used (LRU) cache called the inodes hints cache. We use the inode hint cache to resolve the frequently used file paths using single-batch query. The inodes cache does not require cache coherency mechanisms to keep the inode cache consistent across the namenodes because if the inodes hints cache contains invalid primary keys, then the batch operation would fail and then the namenode will fall back to recursively resolving the file path using primary key operations. After recursively reading the path components from the database, the namenodes also update the inode cache. Additionally, only the move operation changes the primary key of an inode. Move operations are very tiny fraction of the production file system workloads, such as at Spotify less than 2% of the file system operations are file move operations.

Inode Operations

Inode operations go through the following phases.

Pre-transaction Phase

- The namenode receives a file system operation from the client.
- The namenode checks the local inode cache, and if the file path components are found in the cache, then it creates a batch operation to read all file components.

Transactional File System Operation

- The inodes cache is also used to set the partition key hint for the transactional file system operation. The inode ID of the last valid path component is used as partition key hint.
- The transaction is enlisted on the desired NDB datanode according to the partition hint. If the

namenode is unable to determine the partition key hint from the inode cache, then the transaction is enlisted on a random NDB datanode.

- The batched primary key operation is sent to the database to read all the file path inodes. If the batched primary key operation fail due to invalid primary key(s), then the file path is recursively resolved. After resolving the file path components, the namenode also updates the inode cache for future operations.
- The last inode in the file path is read using appropriate lock, that is, read-only file system operation takes shared lock on the inode, while file system operations that need to update the inode take exclusive lock on the inode.
- Depending on the file system operation, some or all of the secondary metadata for the inode is read using partition-pruned index scan operations. The secondary metadata for an inode includes quota, extended metadata, leases, small files data blocks, blocks, replicas, etc., as shown in the entity relational diagram of the file system; see Fig. 5.
- After all the data has been read from the database, the metadata is stored in a per-transaction cache. The file system operation is performed, which reads and updates the data stored in the per-transaction cache. All the changes to the metadata are also stored in the per-transaction cache.
- After the file system operations successfully finish, all the metadata updates are sent to the database to persist the changes for later operations.
- If there are no errors, then the transaction is committed; otherwise the transaction is rolled back.

Post Transaction

- After the transactional file system operation finishes, the results are sent back to the client.
- Additionally, the namenode also updates its matrices about different statistics of the file system operations, such as frequency of different types of file system operations and duration of the operations.

Subtree Operations

Recursive file system operations that operate on large directories are too large to fit in a single database transaction, for example, it is not possible to delete hundreds of millions of files in a single database transaction. HopsFS uses subtree operations protocol to perform such large file system operations. The subtree operations protocol uses application-level locking to mark and isolate inodes. HopsFS subtree operations are designed in such a way that the consistency and operational semantics of the file system operations remain compatible with HDFS. The subtree operations execute in three phases as described below.

Phase 1 – Locking the Subtree: In the first phase, the directory is isolated using application-level locks. Using an inode operation, an exclusive lock is obtained for the directory, and the subtree lock flag is set for the inode. The subtree lock flag also contains other information, such as the ID of the namenode that holds the lock, the type of the file system operation, and the full path of the directory. The flag is an indication that all the descendants of the subtree are locked with exclusive (write) lock. It is important to note that during path resolution, inode and subtree operations that encounter an inode with a subtree lock turned on voluntarily abort the transaction and wait until the subtree lock is removed.

Phase 2 – Quiescing the Subtree: Before the subtree operation updates the descendants of the subtree, it is imperative to ensure that none of the descendants are being locked by any other concurrent subtree or inode operation. Setting the subtree lock before checking if any other descendant directory is also locked for a subtree operation can result in multiple active subtree operations on the same inodes, which will compromise the consistency of the namespace. We store all active subtree operations in a table and query it to ensure that no subtree operations are executing at lower levels of the subtree. In a typical workload, this table does not grow too large as subtree operations are usually only a tiny fraction of all file system operations. Additionally, we wait for all ongoing inode operations to complete by taking and releasing database write

locks on all inodes in the subtree in the same total order used to lock inode. This is repeated down the subtree to the leaves, and a tree data structure containing the inodes in the subtree is built in memory at the namenode. The tree is later used by some subtree operations, such as *move* and *delete* operations, to process the inodes. If the subtree operations protocol fails to quiesce the subtree due to concurrent file system operations on the subtree, it is retried with exponential backoff.

Phase 3 – Executing the FS Operation: Finally, after locking the subtree and ensuring that no other file system operation is operating on the same subtree, it is safe to perform the requested file system operation. In the last phase, the subtree operation is broken into smaller operations, which are run in parallel to reduce the latency of the subtree operation. For example, when deleting a large directory, starting from the leaves of the directory subtree, the contents of the directory is deleted in batches of multiple files.

Handling Failed Subtree Operation

As the locks for subtree operations are controlled by the application (i.e., the namenodes), therefore, it is important that the subtree locks are cleared when a namenode holding the subtree lock fails. HopsFS uses lazy approach for cleaning the subtree locks left by the failed namenodes. The subtree lock flag also contains information about the namenode that holds the subtree lock. Using the group membership service, the namenode maintains a list of all the active namenodes in the system. During the execution of the file system operations, when a namenode encounters an inode with a subtree lock flag set that does not belong to any live namenode, then the subtree operation flag is reset.

It is imperative that the failed subtree operations do not leave the file system metadata in an inconsistent state. For example, the deletion of the subtree progresses from the leaves to the root of the subtree. If the recursive delete operation fails, then the inodes of the partially deleted subtree remain connected to the namespace. When a subtree operation fails due to namenode failure,

then the client resubmits the operation to another alive namenode to complete the file system operation.

Storing Small Files in the Database

HopsFS supports two file storage layers, in contrast to the single file storage service in HDFS, that is, HopsFS can store data blocks on HopsFS datanodes as well as in the distributed database. Large files blocks are stored on the HopsFS datanodes, while small files (≤ 64 KB) are stored in the distributed database. The technique of storing the data blocks with the metadata is also called inode stuffing. In HopsFS, an average file requires 1.5 KB of metadata. As a rule of thumb, if the size of a file is less than the size of the metadata (in our case 1 KB or less), then the data block is stored in memory with the metadata. Other small files are stored in on-disk data tables. The latest solid-state drives (SSDs) are recommended for storing small files data blocks as typical workloads produce large number of random reads/writes on disk for small amounts of data.

Inode stuffing has two main advantages. First, it simplifies the file system operations protocol for reading/writing small files, that is, many network round trips between the client and datanodes are avoided, significantly reducing the expected latency for operations on small files. Second, it reduces the number of blocks that are stored on the datanode and reduces the block reporting traffic on the namenode. For example, when a client sends a request to the namenode to read a file, the namenode retrieves the file's metadata from the database. In case of a small file, the namenode also fetches the data block from the database. The namenode then returns the file's metadata along with the data block to the client. Compared to HDFS, this removes the additional step of establishing a validated, secure communication channel with the datanodes (Kerberos, TLS/SSL sockets, and a block token are all required for secure client-datanode communication), resulting in lower latencies for file read operations (Salman Niazi et al. 2017).

Extended Metadata

File systems provide basic attributes for files/directories such as name, size, modification time, etc. However, in many cases, users and administrators require the ability to tag files/directories with extended attributes for later use. Moreover, rich metadata has become an invaluable feature especially with the continuous growth of data volumes stored in scale-out file systems, such as HopsFS. As discussed earlier HopsFS store all the file system metadata in a database, where each file/directory (inode) is identified by a primary key. Therefore, creating an extended metadata table with a foreign key to the inodes table should be sufficient to ensure consistency and integrity of the metadata and the extended metadata.

HopsFS offers two modes to store extended metadata, that is, *schema-based* and *schemaless* extended metadata. In the schema-based approach, users have to create a predefined schema for their extended metadata, similar to information schema in database systems, and then associate their data according to that schema. The predefined schema enables validation of the extended metadata before attaching them to the inodes. On the other hand, in the schemaless approach, users don't have to create a predefined schema beforehand; instead, they can attach any arbitrary extended metadata that is stored in a self-contained manner such as a JSON file.

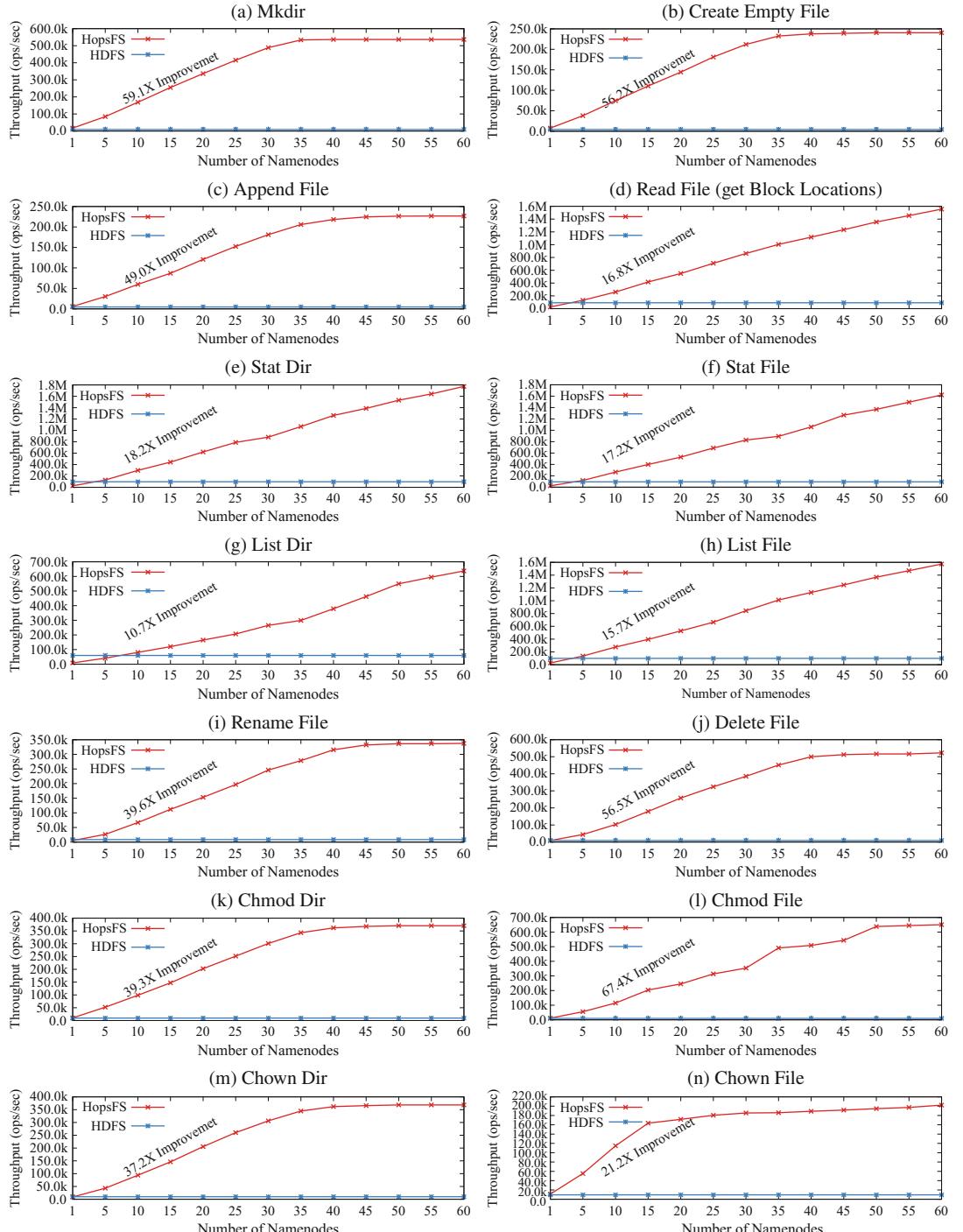
Using either approach schema-based or schemaless, users and administrators can tag their files/directories with extended metadata and then later query for the files/directories based on the tags provided. For example, users can tag files with a description field and later search for the files that match (or partially match) a given free-text query. Also, administrators could use the basic attributes, for example, file size to list all the big files in the cluster, or modification time to list all files that have not been modified within the last month. Depending on the queries pattern, it might be efficient for some queries to run directly on the metadata database, and for others especially the free-text queries, to run on a specialized free-text search engines. Also, the conventional wisdom in database community has been that

there is no-size-fits-all solution. This encourages the use of different storage engines according to the query pattern needed that is known as a polyglot persistence of the data. Therefore, the metadata of the namespace should be replicated into different engines to satisfy different query requirements such as free-text, point, range, and sophisticated join queries. We have developed a databus system, called *ePipe* (Ismail et al. 2017), that provides a strongly consistent replicated metadata service as an extension to HopsFS and that in real-time delivers file system metadata and extended metadata to different storage engines. Users and administrators can later query the different storage engines according to their query requirements.

Results

As HopsFS addresses how to scale out the metadata layer of HDFS, all our experiments are designed to comparatively test the performance and scalability of the namenode(s) in HDFS and HopsFS. Most of the available benchmarking utilities for Hadoop are MapReduce-based, primarily designed to determine the performance of a Hadoop cluster. These benchmarks are very sensitive to the MapReduce subsystem and therefore are not suitable for testing the performance and scalability of namenodes in isolation (Noll 2015). Inspired by the NNThroughputBenchmark for HDFS and the benchmark utility designed to test QFS (Ovsianikov et al. 2013), we developed a distributed benchmark that spawns tens of thousands of file system clients, distributed across many machines, which concurrently execute file system (metadata) operations on the namenode(s). The benchmark utility can test the performance of both individual file system operations and file system workloads based on industrial workload traces. The benchmark utility is open source (Hammer-Bench 2016) and the experiments described here can be fully reproduced on AWS using Karamel and the cluster definitions found in Hammer-Bench (2016).

We ran all the experiments on premise using Dell PowerEdge R730xd servers (Intel(R)



HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases, Fig. 6 Throughput of different file system operations in HopsFS and HDFS. As HDFS only supports only one active namenode, the throughput for HDFS file system operations are represented by the

horizontal (blue) lines. (a) Mkdir. (b) Create empty file. (c) Append file. (d) Read file (getBlockLocations). (e) Stat dir. (f) Stat file. (g) List dir. (h) List file. (i) Rename file. (j) Delete file. (k) Chmod dir. (l) Chmod file. (m) Chown dir. (n) Chown file

Xeon(R) CPU E5-2620 v3 @ 2.40 GHz, 256 GB RAM, 4 TB 7200 RPM HDDs) connected using a single 10 GbE network adapter. Unless stated otherwise, NDB, version 7.5.3, was deployed on 12 nodes configured to run using 22 threads each, and the data replication degree was 2. All other configuration parameters were set to default values. Moreover, in our experiments Apache HDFS, version 2.7.2, was deployed on 5 servers. We did not colocate the file system clients with the namenodes or with the database nodes. As we are only evaluating metadata performance, all the tests created files of zero length (similar to the NNThroughputBenchmark Shvachko 2010). Testing with non-empty files requires an order of magnitude more HDFS/HopsFS datanodes, and provides no further insight.

Figure 6 shows the throughput of different file system operations as a function of number of namenodes in the system. HDFS supports only one active namenode; therefore, the throughput of the file system is represented by (blue) horizontal line in the graphs. From the graphs it is quite clear that HopsFS outperforms HDFS for all file system operations and has significantly better performance than HDFS for the most common file system operations. For example, HopsFS improves the throughput of create file and read file operations by $56.2X$ and $16.8X$, respectively.

Additional experiments for industrial workloads, write intensive synthetic workloads, metadata scalability, and performance under failures are available in HopsFS papers (Niazi et al. 2017; Ismail et al. 2017).

Conclusions

In this entry, we introduced HopsFS, an open-source, highly available, drop-in alternative for HDFS that provides highly available metadata that scales out in both capacity and throughput by adding new namenodes and database nodes. We designed HopsFS file system operations to be deadlock-free by meeting the two preconditions for deadlock freedom: all operations follow the

same total order when taking locks, and locks are never upgraded. Our architecture supports a pluggable database storage engine, and other NewSQL databases could be used, providing they have good support for cross-partition transactions and techniques such as partition-pruned index scans. Most importantly, our architecture now opens up metadata in HDFS for tinkering: custom metadata only involves adding new tables and cleaner logic at the namenodes.

Cross-References

- ▶ [Distributed File Systems](#)
- ▶ [In-memory Transactions](#)
- ▶ [Hadoop](#)

References

- Abad CL (2014) Big data storage workload characterization, modeling and synthetic generation. PhD thesis, University of Illinois at Urbana-Champaign
- Hammer-Bench (2016) Distributed metadata benchmark to HDFS. <https://github.com/smkniazi/hammer-bench>. [Online; Accessed 1 Jan 2016]
- Ismail M, Gebremeskel E, Kakantousis T, Berthou G, Dowling J (2017) Hopsworks: improving user experience and development on hadoop with scalable, strongly consistent metadata. In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS), pp 2525–2528
- Ismail M, Niazi S, Ronström M, Haridi S, Dowling J (2017) Scaling HDFS to more than 1 million operations per second with HopsFS. In: Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing, CCGrid '17. IEEE Press, Piscataway, pp 683–688
- Niazi S, Haridi S, Dowling J (2017) Size matters: improving the performance of small files in HDFS. <https://eurosys2017.github.io/assets/data/posters/poster09-Niazi.pdf>. [Online; Accessed 30 June 2017]
- Niazi S, Ismail M, Haridi S, Dowling J, Grohsschmiedt S, Ronström M (2017) Hopsfs: scaling hierarchical file system metadata using newsql databases. In: 15th USENIX conference on file and storage technologies (FAST'17). USENIX Association, Santa Clara, pp 89–104
- Noll MG (2015) Benchmarking and stress testing an hadoop cluster with TeraSort. TestDFSIO & Co. <http://www.michael-noll.com/blog/2011/04/09/benchmarking->

- [and-stress-testing-an-hadoop-cluster-with-terasort-testdfcio-nnbench-mrbench/](#). [Online; Accessed 3 Sept 2015]
- Ovsianikov M, Rus S, Reeves D, Sutter P, Rao S, Kelly J (2013) The quantcast file system. Proc VLDB Endow 6(11):1092–1101
- Patil SV Gibson GA Lang S, Polte M (2007) GIGA+: scalable directories for shared file systems. In: Proceedings of the 2nd international workshop on petascale data storage: held in conjunction with supercomputing '07, PDSW '07. ACM, New York, pp 26–29
- Ren K, Kwon Y, Balazinska M, Howe B (2013) Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. Proc VLDB Endow 6(10):853–864
- Salman Niazi GB, Ismail M, Dowling J (2015) Leader election using NewSQL systems. In: Proceeding of DAIS 2015. Springer, pp 158–172
- Shvachko KV (2010) HDFS scalability: the limits to growth. Login Mag USENIX 35(2):6–16
- Thomson A, Abadi DJ (2015) CalvinFS: consistent WAN replication and scalable metadata management for distributed file systems. In: 13th USENIX conference on file and storage technologies (FAST 15). USENIX Association, Santa Clara, pp 1–14

HTAP

- ▶ [Hybrid OLTP and OLAP](#)

Hybrid Data Warehouse

- ▶ [Hybrid Systems Based on Traditional Database Extensions](#)

Hybrid DBMS-Hadoop Solution

- ▶ [Hybrid Systems Based on Traditional Database Extensions](#)

Hybrid OLTP and OLAP

Jana Giceva¹ and Mohammad Sadoghi²

¹Department of Computing, Imperial College London, London, UK

²University of California, Davis, CA, USA

H

Synonyms

HTAP; Hybrid transactional and analytical processing; Operational analytics; Transactional analytics

Definitions

Hybrid transactional and analytical processing (HTAP) refers to system architectures and techniques that enable modern database management systems (DBMS) to perform real-time analytics on data that is ingested and modified in the transactional database engine. It is a term that was originally coined by Gartner where Pezzini et al. (2014) highlight the need of enterprises to close the gap between analytics and action for better business agility and trend awareness.

Overview

The goal of running transactions and analytics on the same data has been around for decades, but has not fully been realized due to technology limitations. Today, businesses can no longer afford to miss the real-time insights from data that is in their transactional system as they may lose competitive edge unless business decisions are made on *latest data* (Analytics on *latest data* implies allowing the query to run on any desired level of isolations including dirty read, committed read, snapshot read, repeatable read, or serializable.) or *fresh data* (Analytics on *fresh data* implies running queries on a recent snapshot of data that may not necessarily be the latest possible snapshot when the query execution began or a

consistent snapshot.). As a result, in recent years in both academia and industry, there has been an effort to address this problem by designing techniques that combine the transactional and analytical capabilities and integrate them in a single hybrid transactional and analytical processing (HTAP) system.

Online transaction processing (OLTP) systems are optimized for write-intensive workloads. OLTP systems employ data structures that are designed for high volume of point access queries with a goal to maximize throughput and minimize latency. Transactional DBMSs typically store data in a row format, relying on indexes and efficient mechanisms for concurrency control.

Online analytical processing (OLAP) systems are optimized for heavy read-only queries that touch large amounts of data. The data structures used are optimized for storing and accessing large volumes of data to be transferred between the storage layer (disk or memory) and the processing layer (e.g., CPUs, GPUs, FPGAs). Analytical DBMSs store data in column stores with fast scanning capability that is gradually eliminating the need for maintaining indexes. Furthermore to keep high performance and to avoid the overhead of concurrency, these systems only batch updates at predetermined intervals. This, however, limits the data freshness visible to analytical queries.

Therefore, given the distinct properties of transactional data and (stale) analytical data, most enterprises have opted for a solution that separates the management of transactional and analytical data. In such a setup, analytics is performed as part of a specialized decision support system (DSS) in isolated data warehouses. The DSS executes complex long running queries on data at rest and the updates from the transactional database are propagated via an expensive and slow extract-transform-load (ETL) process. The ETL process transforms data from transactional-friendly to analytics-friendly layout, indexes the data, and materializes selected pre-aggregations. Today's industry requirements are in conflict with such a design. Applications want to interface with fewer systems and try to avoid the burden of moving the transactional data with the expensive ETL process to the analytical

warehouse. Furthermore, systems try to reduce the amount of data replication and the cost that it brings. More importantly, enterprises want to improve data freshness and perform analytics on operational data. Ideally, systems should enable applications to immediately react on facts and trends learned by posing an analytical query within the same transactional request. In short, the choice of separating OLTP and OLAP is becoming obsolete in light of exponential increase in data volume and velocity and the necessity to enrich the enterprise to operate based on real-time insights.

In the rest of this entry, we discuss design decisions, key research findings, and open questions in the database community revolving around exciting and imminent HTAP challenges.

Key Research Findings

Among existing HTAP solutions, we differentiate between two main design paradigms: (1) operating on a single data representation (e.g., row or column store format) vs. multiple representations of data (e.g., both row and column store formats) and (2) operating on a single copy of data (i.e., single data replica) vs. maintaining multiple copies of data (through a variety of data replication techniques).

Examples of systems that operate on a single data representation are SAP HANA by Sikka et al. (2012), HyPer by Neumann et al. (2015), and L-Store by Sadoghi et al. (2016a,b, 2013, 2014). They provide a unified data representation and enable users to analyze the latest (not just fresh data) or any version of data. Here we must note that operating on a single data representation does not necessarily mean avoiding the need for data replication. For instance, the initial version of HyPer relied on a copy-on-write mechanism from the underlying operating system, while SAP HANA keeps the data in a main and a delta, which contains the most recent updates still not merged with the main. L-Store maintains the history of changes to the data (the delta) alongside the lazily maintained latest copy of data in a unified representation form.

Other systems follow the more classical approach of data warehousing, where multiple data representations are exploited, either within a single instance of the data or by dedicating a separate replica for different workloads. Example systems are SQL Server's column-store index proposed by Larson et al. (2015), Oracle's in-memory dual-format by Lahiri et al. (2015), SAP HANA SOE's architecture described by Goel et al. (2015), and the work on fractured mirrors by Ramamurthy et al. (2002), to name a few.

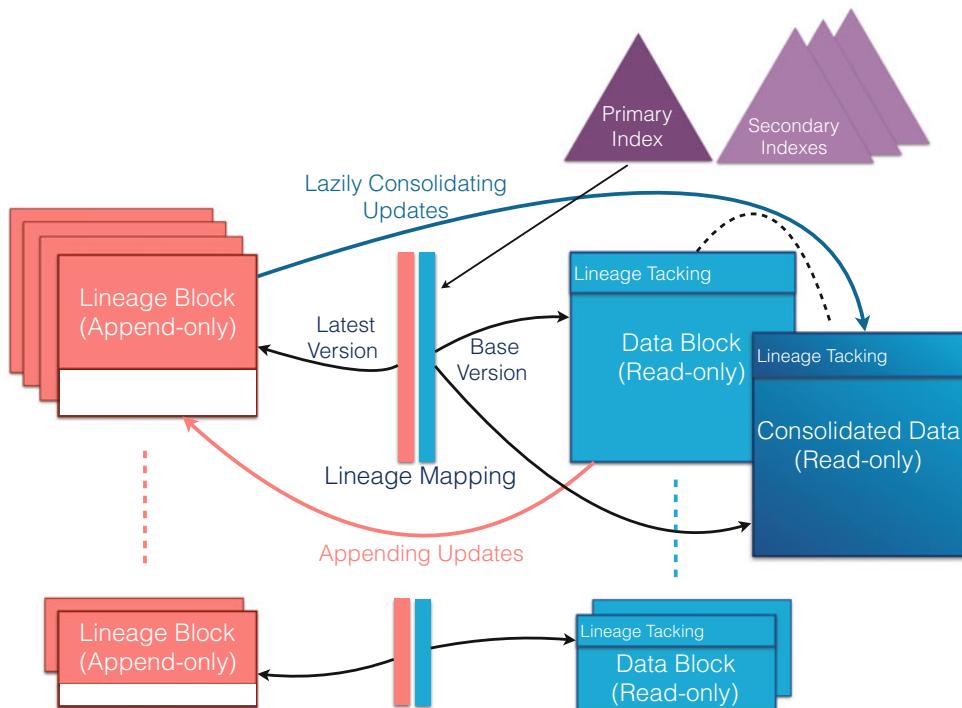
Unified Data Representation

The first main challenge for single data representation HTAP systems is choosing (or designing) a suitable data storage format for hybrid workloads. As noted earlier, there are many systems whose storage format was optimized to support efficient execution of a particular class of workloads. Two prominent example formats are row store (NSM) and column store (DSM), but researchers have also shown the benefit of alternative stores like *partition attribute across* (PAX) proposed by Ailamaki et al. (2001) that strike a balance between the two extremes. OLTP engines typically use a row-store format with highly tuned data structures (e.g., lock-free skip lists, latch-free BW trees), which provide low latencies and high throughput for operational workloads. OLAP engines have been repeatedly shown to perform better if data is stored in column stores. Column stores provide better support for data compression and are more optimized for in-memory processing of large volumes of data. Thus, the primary focus was on developing new storage layouts that can efficiently support both OLTP and OLAP workloads.

The main challenge is the conflict in the properties of data structures suitable to handle large volume of update-heavy requests on the one hand and fast data scans on the other hand. Researchers have explored where to put the updates (whether to apply them in-place, append them in a log-structured format, or store them in a delta and periodically merge them with the main), how to best arrange the records (row-store, column-store, or a PAX format), and how to handle multiple versions and when to do garbage collection.

Sadoghi et al. (2016a,b, 2013, 2014) proposed L-Store (lineage-based data store) that combines the real-time processing of transactional and analytical workloads within a single unified engine. L-Store bridges the gap between managing the data that is being updated at a high velocity and analyzing a large volume of data by introducing a novel update-friendly lineage-based storage architecture (LSA). This is achieved by developing a contention-free and lazy staging of data from write-optimized into read-optimized form in a transactionally consistent manner without the need to replicate data, to maintain multiple representation of data, or to develop multiple loosely integrated engines that limit real-time capabilities.

The basic design of LSA consists of two core ideas, as captured in Fig. 1. First, the base data is kept in read-only blocks, where a block is an ordered set of objects of any type. The modification to the *data blocks* (also referred to as *base pages*) is accumulated in the corresponding *lineage blocks* (also referred to as *tail pages*). Second, a lineage mapping links an object in the data blocks to its recent updates in a lineage block. This essentially decouples the updates from the physical location of objects. Therefore, via the lineage mapping, both the base and updated data are retrievable. A lazy, contention-free background process merges the recent updates with their corresponding read-only base data in order to construct new consolidated data blocks while data is continuously being updated in the foreground without any interruption. The merge is necessary to ensure optimal performance for the analytical queries. Furthermore, each data block internally maintains the lineage of the updates consolidated thus far. By exploiting the decoupling and lineage tracking, the merge process, which only creates a new set of read-only consolidated data blocks, is carried out completely independently from update queries, which only append changes to lineage blocks and update the lineage mapping. Hence, there is no contention in the write paths of update queries and the merge process. That is a fundamental property necessary to build a highly scalable distributed storage layer that is updatable.



Hybrid OLTP and OLAP, Fig. 1 Lineage-based storage architecture (LSA)

Neumann et al. (2015) proposed a novel MVCC implementation, which does update in place and store prior versions as before-image deltas, which enable both an efficient scan execution and fine-grained serializability validation needed for fast processing of point access transactions. From the NoSQL side, Pilman et al. (2017) demonstrated how scans can be efficiently implemented on a key-value store (KV store) enabling more complex analytics to be done on large and distributed KV stores. Wildfire from IBM by Barber et al. (2016, 2017) uses Apache Parquet to support both analytical and transactional requests. IBM Wildfire is a variant of IBM DB2 BLU that is integrated into Apache Spark to support fast ingest of updates. The authors adopt the relaxed last-writer-wins semantics and offer an efficient snapshot isolation on recent view of data (but stale) by relying on periodic shipment and writing of the logs onto a distributed file system.

Manifold of Data Representations

An alternative approach comes from systems that propose using hybrid stores to keep the data in two or more layouts, either by partitioning the data based on the workload properties or by doing partial replication. They exploit manifold specialization techniques that are aligned with the tenet “one size does not fit all” as explained by Stonebraker and Cetintemel (2005). Typically the hybrid mode consists of storing data in row and column formats by collocating attributes that are accessed together within a query. The main differentiators among the proposed solutions come by addressing the following challenges: How does a system determine which data to store in which layout – is it specified manually by the DB administrator, or is it derived by the system itself? Is the data layout format static or can it change over time? If the latter, which data is affected by the transformations – is it the hot or cold data? Does the system support update propagation or

is the data transformation only recommended for read-only data? If the former, when does the data transformation take place – is it incremental or immediate? Does it happen as part of query execution or is it done as part of a background process?

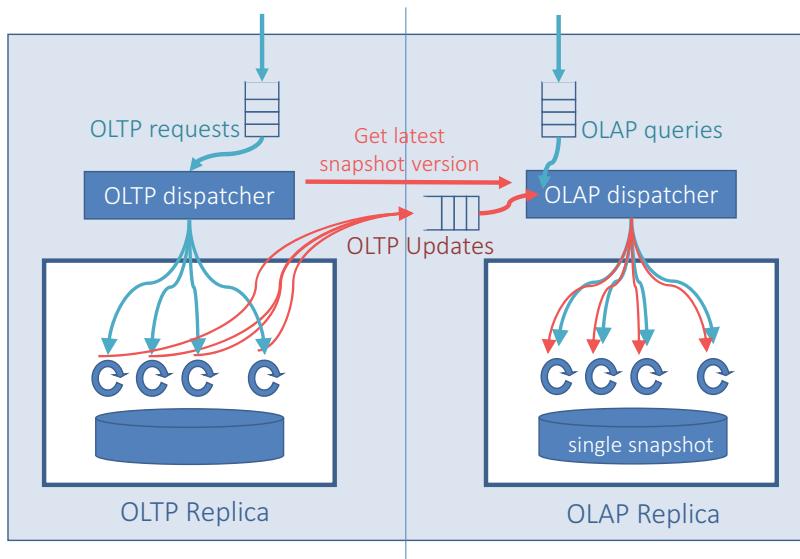
Grund et al. (2010) with their HYRISE system propose the use of the *partially decomposed* storage model and automatically partition tables into vertical partitions of varying depth, depending on how the columns are accessed. Unfortunately, the bandwidth savings achieved with this storage model come with an increased CPU cost. A follow-up work by Pirk et al. (2013) improves the performance by combining the partially decomposed storage model with just-in-time (JiT) compilation of queries, which eliminates the CPU-inefficient function calls. Based on the work in HYRISE, Plattner (2009) in SAP developed HANA, where tuples start out in a row-store and are then migrated to a compressed column-store storage manager. Similarly, also MemSQL, Microsoft's SQL Server, and Oracle support the two storage formats. One of the main differentiators among these systems is who decides on the partitioned layout: i.e., whether the administrator manually specifies which relations and attributes should be stored as a column store or the system derives it automatically. The latter can be achieved by monitoring and identifying hot vs. cold data or which attributes are accessed together in OLAP queries. There are a few research systems that have demonstrated the benefits of using adaptive data stores that can transform the data from one format to another with an evolving HTAP workload. Dittrich and Jindal (2011) show how to maintain multiple copies of the data in different layouts and use a logical log as a primary storage structure before creating a secondary physical representation from the log entries. In H2O, Alagiannis et al. (2014) maintain the same data in different storage formats and improve performance of the read-only workload by leveraging multiple processing engines. The work by Arulraj et al. (2016) performs data reorganization on cold data. The

reorganization is done as part of an incremental background process that does not impact the latency-sensitive transactions but improves the performance of the analytical queries.

Data Replication

Operating on a single data replica may come at a cost due to an identified trade-off between degree of data freshness, system performance, and predictability of throughput and response time for a hybrid workload mix. Thus, the challenges for handling HTAP go beyond the layout in which the data is stored. A recent study by Psaroudakis et al. (2014) shows that systems like SAP HANA and HyPer, which rely on partial replication, experience interference problems between the transactional and analytical workloads. The observed performance degradation is attributed to both resource sharing (physical cores and shared CPU caches) and synchronization overhead when querying and/or updating the latest data.

This observation motivated researchers to revisit the benefits of data replication and address the open challenge of efficient update propagation. One of the first approaches for addressing hybrid workloads was introduced by Ramamurthy et al. (2002), where the authors proposed replicating the data and storing it in two data formats (row and columnar). The key advantage is that the data layouts and the associated data structures as well as the data processing techniques can be tailored to the requirements of the particular workload. Additionally, the system can make efficient use of the available hardware resources and their specialization. This is often viewed as a loose-form of an HTAP system. The main challenge for this approach is maintaining the OLAP replica up to date and avoiding the expensive ETL process. BatchDB by Makreshanski et al. (2017) relies on primary-secondary form of replication with the primary replica handling the OLTP workload and the updates being propagated to a secondary replica that handles the OLAP workload (Fig. 2). BatchDB successfully addresses the problem of performance interference by spatially partitioning resources to the



Hybrid OLTP and OLAP, Fig. 2 Replication-based HTAP architecture

two data replicas and their execution engines (e.g., either by allocating a dedicated machine and connecting the replicas with RDMA over InfiniBand or by allocating separate NUMA nodes within a multi-socket server machine). The system supports high-performance analytical query processing on fresh data with snapshot isolation guarantees. It achieves that by queuing queries and updates on the OLAP side and scheduling them in batches, executing one batch at a time. BatchDB ensures that the OLAP queries see the latest snapshot of the data by using a lightweight propagation algorithm, which incurs a small overhead on the transactional engine to extract the updates and efficiently applies them on the analytical side at a faster rate than what has been recorded as a transactional throughput in literature so far. Such a system design solves the problem of high performance and isolation for hybrid workloads, and most importantly enables data analytics on fresh data. The main problem is that it limits the functionality of what an HTAP application may do with the data. For instance, it does not support interleaving of analytical queries within a transaction and does not allow analysts to go back in time and explore prior versions of the data to explain certain trends.

Oracle GoldenGate and SAP HANA SOE also leverage data replication and rely on specialized

data processing engines for the different replicas. In GoldenGate, Oracle uses a publish/subscribe mechanism and propagates the updates from the operational database to the other replicas if they have subscribed to receive the updates log.

Examples of Applications

There are many possible applications of HTAP for modern businesses. With today's dynamic data-driven world, real-time advanced analytics (e.g., planning, forecasting, and what-if analysis) becomes an integral part of many business processes rather than a separate activity after the events have happened. One example is online retail. The transactional engine keeps track of data including the inventory list and products and the registered customers and manages all the purchases. An analyst can use this online data to understand customer behavior and come up with better strategies for product placement, optimized pricing, discounts, and personalized recommendations as well as to identify products which are in high demand and do proactive inventory refill. Another business use for HTAP comes from the financial industry sector. The transactional engine already supports millions of banking transactions per second and real-time fraud detection and action which can save billions of dollars. Yet

another example of real-time detection and action is inspired by content delivery networks (CDNs), where businesses need to monitor the network of web servers that deliver and distribute the content at real time. An HTAP system can help identify distributed denial-of-service (DDoS) attacks, find locations with spikes to redistribute the traffic bandwidth, get the top- k customers in terms of traffic usage, and be able to act on it immediately.

Future Directions of Research

At the time of writing this manuscript, we argue that there is yet any system or technique that fulfill all HTAP promises to successfully interleave analytical processing within transactions. This is sometimes referred to as *true* or *in-process* HTAP, i.e., real-time analytics that is not limited to latest committed data and posed as a read-only request but incorporates the analytics as part of the same write-enabled transaction request. Supporting *in-process* HTAP is an ongoing effort in the research community and much sought-after feature for many enterprise systems.

Another open question is to investigate whether HTAP systems could and should support different types of analytical processing in addition to traditional OLAP analytics, e.g., the emergence of polystore by Duggan et al. (2015). Training and inferring from machine learning models that describe fraudulent user behavior could supplement the existing knowledge of OLAP analytics. Furthermore, with the support of graph processing, one can finally uncover fraudulent rings and other sophisticated scams that can be best identified using graph analytics queries. Therefore, it would be of great use if the graph analytical engine can process online data and be integrated in a richer and more expressive HTAP system (e.g., Hassan et al. 2017).

Finally, there is the question on how hardware heterogeneity could impact the design of HTAP systems. With the end of Moore's Law, the hardware landscape has been shifting towards specialization of computational units (e.g., GPUs, Xeon Phi, FPGAs, near-memory computing, TPUs) (Najafi et al. 2017, 2015; Teubner and Woods 2013). Hardware has always been

an important game changer for databases, and as in-memory processing enabled much of the technology for approaching true HTAP, there is discussion that the coming computing heterogeneity is going to significantly influence the design of future systems as highlighted by Appuswamy et al. (2017).

Cross-References

- ▶ Active Storage
- ▶ Blockchain Transaction Processing
- ▶ Hardware-Assisted Transaction Processing: NVM
- ▶ In-memory Transactions

References

- Ailamaki A, DeWitt DJ, Hill MD, Skounakis M (2001) Weaving relations for cache performance. In: VLDB, pp 169–180
- Alagiannis I, Idreos S, Ailamaki A (2014) H2O: a hands-free adaptive store. In: SIGMOD, pp 1103–1114
- Appuswamy R, Karpathiotakis M, Porobic D, Ailamaki A (2017) The case for heterogeneous HTAP. In: CIDR
- Arulraj J, Pavlo A, Menon P (2016) Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: SIGMOD, pp 583–598
- Barber R, Huras M, Lohman G, Mohan C, Mueller R, Özcan F, Pirahesh H, Raman V, Sidle R, Sidorkin O, Storm A, Tian Y, Tözün P (2016) Wildfire: concurrent blazing data ingest and analytics. In: SIGMOD'16, pp 2077–2080
- Barber R, Garcia-Arellano C, Grosman R, Müller R, Raman V, Sidle R, Spilchen M, Storm AJ, Tian Y, Tözün P, Zilio DC, Huras M, Lohman GM, Mohan C, Özcan F, Pirahesh H (2017) Evolving databases for new-gen big data applications. In: Online Proceedings of CIDR
- Dittrich J, Jindal A (2011) Towards a one size fits all database architecture. In: CIDR
- Duggan J, Elmore AJ, Stonebraker M, Balazinska M, Howe B, Kepner J, Madden S, Maier D, Mattson T, Zdonik S (2015) The BigDAWG polystore system. SIGMOD Rec 44(2):11–16
- Goel AK, Pound J, Auch N, Bumbulis P, MacLean S, Färber F, Gropengiesser F, Mathis C, Bodner T, Lehner W (2015) Towards scalable real-time analytics: an architecture for scale-out of OLxP workloads. PVLDB 8(12):1716–1727
- Grund M, Krüger J, Plattner H, Zeier A, Cudré-Mauroux P, Madden S (2010) HYRISE – a main memory hybrid storage engine. In: PVLDB, pp 105–116
- Hassan MS, Kuznetsova T, Jeong HC, Aref WG, Sadoghi M (2017) Empowering in-memory relational

- database engines with native graph processing. CoRR abs/1709.06715
- Lahiri T, Chavan S, Colgan M, Das D, Ganesh A, Gleeson M, Hase S, Holloway A, Kamp J, Lee TH, Loaiza J, Macnaughton N, Marwah V, Mukherjee N, Mullick A, Muthulingam S, Raja V, Roth M, Soylemez E, Zait M (2015) Oracle database in-memory: a dual format in-memory database. In: ICDE, pp 1253–1258. <https://doi.org/10.1109/ICDE.2015.7113373>
- Larson PA, Birka A, Hanson EN, Huang W, Nowakiewicz M, Papadimos V (2015) Real-time analytical processing with SQL server. PVLDB 8(12):1740–1751
- Makreshanski D, Giceva J, Barthels C, Alonso G (2017) BatchDB: efficient isolated execution of hybrid OLTP+OLAP workloads for interactive applications. In: SIGMOD'17, pp 37–50
- Najafi M, Sadoghi M, Jacobsen H (2015) The FQP vision: flexible query processing on a reconfigurable computing fabric. SIGMOD Rec 44(2):5–10
- Najafi M, Zhang K, Sadoghi M, Jacobsen H (2017) Hardware acceleration landscape for distributed real-time analytics: virtues and limitations. In: ICDCS, pp 1938–1948
- Neumann T, Mühlbauer T, Kemper A (2015) Fast serializable multi-version concurrency control for main-memory database systems. In: SIGMOD, pp 677–689
- Pezzini M, Feinberg D, Rayner N, Edjali R (2014) Hybrid transaction/analytical porcessing will foster opportunities for dramatic business innovation. <https://www.gartner.com/doc/2657815/hybrid-transactional-analytical-processing-foster-opportunities>
- Pilman M, Bocksrocker K, Braun L, Marroquín R, Kossmann D (2017) Fast scans on key-value stores. PVLDB 10(11):1526–1537
- Pirk H, Funke F, Grund M, Neumann T, Leser U, Manegold S, Kemper A, Kersten ML (2013) CPU and cache efficient management of memory-resident databases. In: ICDE, pp 14–25
- Plattner H (2009) A common database approach for OLTP and OLAP using an in-memory column database. In: SIGMOD, pp 1–2
- Psaroudakis I, Wolf F, May N, Neumann T, Böhm A, Ailamaki A, Sattler KU (2014) Scaling up mixed workloads: a battle of data freshness, flexibility, and scheduling. In: TPCTC 2014, pp 97–112
- Ramamurthy R, DeWitt DJ, Su Q (2002) A case for fractured mirrors. In: VLDB'02, pp 430–441
- Sadoghi M, Ross KA, Canim M, Bhattacharjee B (2013) Making updates disk-I/O friendly using SSDs. PVLDB 6(11):997–1008
- Sadoghi M, Canim M, Bhattacharjee B, Nagel F, Ross KA (2014) Reducing database locking contention through multi-version concurrency. PVLDB 7(13): 1331–1342
- Sadoghi M, Bhattacharjee S, Bhattacharjee B, Canim M (2016a) L-store: a real-time OLTP and OLAP system. CoRR abs/1601.04084
- Sadoghi M, Ross KA, Canim M, Bhattacharjee B (2016b) Exploiting SSDs in operational multiversion databases. VLDB J 25(5):651–672
- Sikka V, Färber F, Lehner W, Cha SK, Peh T, Bornhövd C (2012) Efficient transaction processing in SAP HANA database: the end of a column store myth. In: SIGMOD'12, pp 731–742
- Stonebraker M, Cetintemel U (2005) “One size fits all”: an idea whose time has come and gone. In: ICDE, pp 2–11
- Teubner J, Woods L (2013) Data processing on FPGAs. Synthesis lectures on data management. Morgan & Claypool Publishers, San Rafael
-
- ## Hybrid Systems Based on Traditional Database Extensions
- Yuanyuan Tian
IBM Research – Almaden, San Jose, CA, USA
- ### Synonyms
- Hadoop EDW Integration; Hybrid DBM-S-Hadoop Solution; Hybrid Data Warehouse; Hybrid Warehouse
- ### Definitions
- A hybrid system based on traditional database extensions refers to a federated system between Hadoop and an enterprise data warehouse (EDW). In such a system, a query may need to combine data stored in both Hadoop and an EDW.
- ### Overview
- The co-existence of Hadoop and enterprise data warehouses (EDWs), together with the new application requirement of correlating data stored in both systems, has created the need for a special federation between Hadoop-like big data platforms and EDWs. This entry presents the motivation behind such hybrid systems, highlights the unique challenges of building them, surveys existing hybrid solutions, and finally discusses potential future directions.

Introduction

More and more enterprises today start to embrace Hadoop-like big data technologies to process huge volumes of data and drive actionable insights. The Hadoop Distributed File System (HDFS) has become the core storage system for enterprise data, including enterprise application data, social media data, log data, click stream data, and other Internet data. Systems that support SQL queries over data stored on HDFS, dubbed as *SQL-on-Hadoop* systems, have emerged over the years, including Hive (Thusoo et al. 2009), Spark SQL (Armbrust et al. 2015), IBM Big SQL (Gray et al. 2015), and Impala (Kornacker et al. 2015).

At the same time, enterprise data warehouses (EDWs) continue to support critical business analytics. EDWs are usually massively parallel processing (MPP) databases that support complex SQL processing, updates, and transactions (EDW and database are used interchangeably in this entry). As a result, they manage up-to-date data and support various business analytics tools, such as reporting and dashboards.

Many new applications have emerged, requiring combining the large volumes of semi-structured and structured, somewhat raw big data, with the well ETLed warehouse data. An example scenario is presented below to illustrate the real application needs.

An Example Application Consider a retailer, such as Walmart or Target, who sells products in local stores as well as online. All the transactions, either offline or online, are managed and stored in a parallel database, whereas users' online click logs are captured and stored in a SQL-on-Hadoop system. The retailer wants to analyze the correlation of customers' online behaviors with sales data. This requires joining the transaction table T in the parallel database with the log table L in the Hadoop. One such analysis can be expressed as the following SQL query.

```
SELECT L.urlPrefix, COUNT(*)
FROM T, L
WHERE T.category = 'Canon Camera'
AND region(L.ip) = 'East Coast'
AND T.uid = L.uid
```

**AND T.tDate >= L.lDate AND T.tDate <= L.lDate + 1
GROUP BY L.urlPrefix**

This query tries to find out the number of views of the URLs visited by customers with IP addresses from the East Coast who bought Canon cameras within 1 day of their online visits.

Applications like the above example, together with the coexistence of Hadoop and EDWs, have created the need for a new generation of special federation between Hadoop and EDWs. The big data industry has used different terms to refer to this special federation, such as "Hadoop EDW integration" (Shankar 2015), "hybrid DBMS-Hadoop solution" (Portnoy 2013), "hybrid data warehouse" (Hortonworks 2015), or simply "hybrid warehouse" (Tian et al. 2015, 2016). This entry chooses to use the term "hybrid warehouse" to refer to this special federation.

The need for interacting with Hadoop has prompted many EDW vendors like Oracle, IBM, Teradata, and Microsoft to develop their hybrid warehouse solutions. Most of them start from their existing relational databases and augment them with the abilities to interoperate with Hadoop. This entry will focus on these hybrid warehouse solutions based on traditional database extensions.

H

Challenges

Before surveying the hybrid warehouse solutions, it is very important to highlight the unique challenges of the hybrid warehouse:

- Although SQL processing capability on Hadoop data has been provided by many SQL-on-Hadoop systems, a full-pledged relational database (EDW) and a SQL-on-Hadoop query processor have very different characteristics. First, the database owns its data, hence controls the partitioning and data organization. This is not true for a SQL-on-Hadoop query processor, which works with existing files on a HDFS. They do not dictate the physical layout of data, because it is simply prohibitive to convert petabytes of data on HDFS to a proprietary format before

processing. This is fundamentally different from databases. Second, a database can have indexes, but a SQL-on-Hadoop processor cannot exploit the same record-level indexes. Because they do not control the data; data can be inserted outside the query processor's control. In addition, HDFS does not provide the same level of performance for small reads as a local file system, as it was mainly designed for large scans. Finally, HDFS does not support update in place; as a result, SQL-on-Hadoop systems are mostly used for processing data that are not or infrequently updated. Although the latest version of Hive has added the transaction support, it is done through deltas, thus more appropriate for batch updates. These differences between the two systems make this problem both hybrid and asymmetric.

- There are also differences in terms of capacity and cluster setup. While EDWs are typically deployed on high-end hardware, Hadoop (or a SQL-on-Hadoop system) is deployed on commodity servers. A Hadoop cluster typically has a much larger scale (up to 10,000s machines), hence has more storage and computational capacity than an EDW. In fact, today more and more enterprise investment has shifted from EDWs to Hadoop.
- Finally, the hybrid warehouse is also not a typical federation between two databases. Existing federation solutions (Josifovski et al. 2002; Adali et al. 1996; Shan et al. 1995; Tomasic et al. 1998; Papakonstantinou et al. 1995) use a client-server model to access the remote databases and move the data. In par-

ticular, they use JDBC/ODBC interfaces for pushing down a maximal sub-query and retrieving its results. Such a solution ingests the result data serially through the single JDBC/ODBC connection and hence is only feasible for small amounts of data. In the hybrid warehouse case, a new solution that connects at a lower system layer is needed to exploit the massive parallelism on both the Hadoop side and the EDW side and move data between the two in parallel. This requires nontrivial coordination and communication between the two systems.

Table 1 summarizes the differences between an EDW and Hadoop, hence highlights the challenges of hybrid warehouse solutions.

Hybrid Warehouse Solutions Based on Database Extensions

As Hadoop ecosystem and related big-data technologies mature, the hybrid warehouse solutions have also evolved through several iterations.

Parallel Movement of Data in Bulk

The first iteration of hybrid warehouse solutions came right after EDW vendors realized that Hadoop is a major disruption to traditional EDWs. Both EDW and Hadoop vendors started to allow data to be moved between EDWs and Hadoop efficiently, and to improve performance most solutions leverage the massive parallelism in Hadoop and EDWs to transfer data in parallel. [Sqoop](#) is such a general tool from

Hybrid Systems Based on Traditional Database Extensions, Table 1 Differences between EDWs and Hadoop

	EDW	Hadoop
Data ownership	Own its data	Work with existing files on shared storage
	Control data organization and partitioning	Cannot dictate data layout
Index support	Build and exploit index	Scan-based only, no index support
Update	Efficient record-level updates	Good for batch updates
	High-end servers	Commodity machines
Capacity	Smaller cluster size	Larger cluster size (up to 10,000 s nodes)

the Hadoop camp that imports database tables onto HDFS in different formats and exports data in HDFS directories as database tables. Many EDW vendors have also provided their own version of “Hadoop connectors.” Özcan et al. (2011) introduced a utility to bulk-load HDFS data to IBM Db2. Teradata Connector for Hadoop (Teradata 2013) supports bi-directional bulk load between Teradata systems and Hadoop ecosystem components, such as HDFS and Hive. HP Vertica Connector for HDFS (Vertica 2016) allows bulk-loading HDFS data into HP Vertica. Oracle Loader for Hadoop (Oracle 2012) bulk-loads data from Hadoop to Oracle Database.

Batch movement of data between EDWs and Hadoop has several limitations. On one hand, with this approach, one data set resides in two places, thus maintaining it requires extra care. In particular, HDFS still does not have a good solution to do update in place, whereas warehouses always have up-to-date data. As a result, an application needs to reload EDW data to Hadoop frequently to keep the data fresh. On the other hand, HDFS tables are usually pretty big, so it is not always feasible to load them into the database. Such bulk reading of HDFS data into the database introduces an unnecessary burden on the carefully managed database resources.

EDWs Directly Reading HDFS Data

To avoid data replication, the second wave of hybrid warehouse solutions keep data where they are and allow EDWs to directly read HDFS data through external tables or table functions. For example, HP Vertica Connector for HDFS (Vertica 2016), besides the bulk-loading approach described in the previous subsection, also supports exposing the HDFS data as external tables to query from. Özcan et al. (2011) proposed a table function called HDFSRead to load HDFS data at runtime to IBM Db2 and a new DB2InputFormat to ingest data from Db2 to Hadoop. Oracle Direct Connector for HDFS (Oracle 2012) directly accesses data on HDFS as external tables from Oracle Database.

In this second iteration of hybrid solutions, the EDWs treat HDFS as an extended storage and pull data when needed from HDFS. However, the query processing is performed only inside the database. This is probably due to the fact that in the early days, people mostly viewed Hadoop as a good option for cheap storage and the processing support on Hadoop was still limited (MapReduce was not easy to program against). But as the first SQL-on-Hadoop system – Hive, as well as other systems exposing high-level language abstraction on top of MapReduce, such as Pig (Olston et al. 2008) and Jaql (Beyer et al. 2011), started to emerge, data processing on Hadoop became easier and more powerful.

H

Query Pushdown to Hadoop

Although the second wave of hybrid solutions do not require data replication, they still load an entire data set at runtime into the database for processing, even though only a small fraction of the data may be needed (e.g., through filter and projection). The processing burden on databases is still very high. Riding on the SQL-on-Hadoop boom, the third wave of hybrid warehouse solutions start to offload some data processing to the Hadoop side, by pushing mostly filters and projections down to Hadoop. Oracle Big Data SQL (McClary 2014) allows Oracle Exadata to query remote data on Hadoop via external tables. But, different from the external table approach in Oracle Direct Connector for HDFS, Oracle Big Data SQL is able to push down filter and projection to the Hadoop side by employing *smart scans* on Hadoop data. Teradata QueryGrid (Teradata 2016) provides a unified data architecture to query data in Teradata and Aster databases as well as other data sources. In particular, it enables a Teradata or Aster databases to call up Hive to query data on Hadoop and bring the results back to Teradata or Aster to join with other tables. Microsoft Polybase (DeWitt et al. 2013) also exposes HDFS data as external tables in SQL Server PDW and pushes filter and projection down to the Hadoop side. In addition, it also considers pushing down joins if both tables are stored in HDFS.

In all these systems, however, a join between an HDFS table and an EDW table is always executed inside the EDW. This is probably due to the assumption that SQL-on-Hadoop systems do not perform joins efficiently. Although this was true for the early SQL-on-Hadoop solutions, such as Hive (Thusoo et al. 2009) before version 0.13, it is not clear whether the same still holds for the current-generation solutions such as IBM Big SQL (Gray et al. 2015), Impala (Kornacker et al. 2015), and Presto (Traverso 2013). There has been a significant shift during the last few years in the SQL-on-Hadoop solution space, in which many new systems have moved away from MapReduce to have shared-nothing parallel database architectures. They run SQL queries using their own long-running daemons executing on every HDFS DataNode. Instead of materializing intermediate results, these systems pipeline them between computation stages. Many new SQL-on-Hadoop systems also employ columnar store technologies, vector-wise processing, and even runtime code generation for better query performance.

Moreover, HDFS tables are usually much bigger than database tables, so it is not always feasible to ingest HDFS data and perform joins in the database. Another important observation is that enterprises are investing more on big data systems such as Hadoop and less on expensive EDW systems. As a result, there is more capacity on the Hadoop side to leverage.

Collaborative Query Processing

Finally, with the advancement in the SQL-on-Hadoop land, it is possible to have EDWs and Hadoop to collaboratively process a query, even including executing a join together.

Tian et al. (2015) studied a number of distributed join algorithms for joining a HDFS table with an EDW table in the context of a hybrid warehouse. Besides algorithms that execute joins inside EDW, this work particularly explores algorithms that conduct the join on the Hadoop side, including a novel join algorithm called *zigzag* join, which heavily exploits the use of Bloom

filter (Bloom 1970) on both sides to minimize the data movement in join processing. Through extensive experimental study, Tian et al. (2015) concluded that with a sophisticated SQL execution engine on the Hadoop side, which most of the new-generation SQL-on-Hadoop systems qualify, it is better to move the smaller table to the side of the bigger table (after filter and projection are applied on the tables), whether it is in HDFS or in the EDW. Moreover, Bloom filter is very effective in reducing data movement in a hybrid warehouse setting, and it can even be applied on both sides in one join algorithm. In a follow-up work, Tian et al. (2016) proposed a cost model of various join algorithms for a future optimizer in the hybrid warehouse environment to choose the right algorithms.

Conclusion and Future Directions

Hadoop has become an important data repository in the enterprise as the center for all business analytics, from SQL queries to machine learning, to reporting. At the same time, EDWs continue to support critical business analytics. EDWs will coexist with Hadoop. That is why there is a strong need to have hybrid warehouse solutions. The hybrid warehouse is different from the traditional data federation. It requires exploiting massive parallelism between two sides for efficient data movement, and better collaborative query processing to minimize data that needs to be moved, especially given the powerful SQL processing capabilities on Hadoop in the recent years. With more and more enterprise investment on the Hadoop side, it is wise to better utilize the resource of the Hadoop side to do more of the processing.

In the future, EDWs and Hadoop will continue to coexist and evolve on their own, and the need for hybrid warehouse solutions will continue to grow. We need an even tighter integration between the two systems, in which an automated optimizer can decide where to store data and where the data processing is carried out, without the users even knowing (Portnoy 2013).

Cross References

- ▶ [Hive](#)
- ▶ [Impala](#)
- ▶ [Spark SQL](#)
- ▶ [Distributed File Systems](#)
- ▶ [Hadoop](#)

References

- Adali S, Seluk Candan K, Papakonstantinou Y, Subrahmanian VS (1996) Query caching and optimization in distributed mediator systems. In: Proceedings of the 1996 ACM SIGMOD international conference on management of data (SIGMOD'96), pp 137–146
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data (SIGMOD'15), pp 1383–1394
- Beyer KS, Ercegovac V, Gemulla R, Balmin A, Eltabakh MY, Kanne C-C, Özcan F, Shekita EJ (2011) Jaql: a scripting language for large scale semistructured data analysis. *PVLDB* 4(12):1272–1283
- Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. *Commun ACM* 13(7):422–426
- DeWitt DJ, Halverson A, Nehme RV, Shankar S, Aguilarsaborit J, Avanes A, Flasza M, Gramling J (2013) Split query processing in Polybase. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data (SIGMOD'13), pp 1255–1266
- Gray SC, Ozcan F, Pereyra H, van der Linden B, Zubiri A (2015) SQL-on-hadoop without compromise: how big SQL 3.0 from IBM represents an important leap forward for speed, portability and robust functionality in SQL-on-hadoop solutions. <http://public.dhe.ibm.com/common/ssi/ecm/sw/en/swv14019usen/SWW14019USEN.PDF>
- Hive (2018) Apache Hive. <https://hive.apache.org/>
- Hortonworks (2015) Modern data architecture with apache hadoop – the hybrid data warehouse. <https://www.denodo.com/en/system/files/document-attachments/wp-hortonworks-01-ab.pdf>
- Josifovski V, Schwarz P, Haas L, Lin E (2002) Garlic: a new flavor of federated query processing for Db2. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data (SIGMOD'02), pp 524–532
- Kornacker M, Behm A, Bittorf V, Bobrovitsky T, Ching C, Choi A, Erickson J, Grund M, Hecht D, Jacobs M, Joshi I, Kuff L, Kumar D, Leblang A, Li N, Pandis I, Robinson H, Rorke D, Rus S, Russell J, Tsirogiannis D, Wanderman-Milne S, Yoder M (2015) Impala: a modern, open-source SQL engine for hadoop. In: Proceedings of the 2015 conference on innovative data systems research (CIDR)
- McClary D (2014) Oracle big data SQL: one fast query, all your data. https://blogs.oracle.com/datawarehousing/entry/oracle_big_data_sql_one
- Olston C, Reed B, Srivastava U, Kumar R, Tomkins A (2008) Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD'08), pp 1099–1110
- Oracle (2012) High performance connectors for load and access of data from Hadoop to Oracle database. <http://www.oracle.com/technetwork/bdc/hadoop-loader/connectors-hdfs-wp-1674035.pdf>
- Özcan F, Hoa D, Beyer KS, Balmin A, Liu CJ, Li Y (2011) Emerging trends in the enterprise data analytics: connecting hadoop and Db2 warehouse. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data (SIGMOD'11), pp 1161–1164
- Papakonstantinou Y, Gupta A, Garcia-Molina H, Ullman JD (1995) A query translation scheme for rapid implementation of wrappers. In: Proceedings of the 1995 international conference on deductive and object-oriented databases (DOOD'95), pp 161–186
- Pig (2018) Apache Pig. <https://pig.apache.org>
- Portnoy D (2013) The future of hybrid data warehouse-Hadoop implementations. <https://www.slideshare.net/DavidPortnoy/hybrid-data-warehouse-hadoop-implementations>
- Shan M-C, Ahmed R, Davis J, Du W, Kent W (1995) Pegasus: a heterogeneous information management system. In: Kim W (ed) Modern database systems. ACM Press/Addison-Wesley Publishing Co., New York/Reading, pp 664–682
- Shankar R (2015) Top reasons for powering your enterprise data warehouse with hadoop. <http://www.cignex.com/blog/top-reasons-powering-your-enterprise-data-warehouse-hadoop>
- SparkSQL (2018) Spark SQL. <https://spark.apache.org/sql>
- Sqoop (2018) Apache Sqoop. <http://sqoop.apache.org>
- Teradata (2013) Teradata connector for Hadoop. <http://developer.teradata.com/connectivity/articles/teradata-connector-for-hadoop-now-available>
- Teradata (2016) Take a giant step with Teradata QueryGrid. <http://blogs.teradata.com/data-points/take-a-giant-step-with-teradata-querygrid>
- Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R (2009) Hive: a warehousing solution over a map-reduce framework. *Proc VLDB Endow (PVLDB)* 2(2):1626–1629
- Tian Y, Zou T, Ozcan F, Goncalves R, Pirahesh H (2015) Joins for hybrid warehouses: exploiting massive parallelism in hadoop and enterprise data warehouses. In: Proceedings of the 2015 international conference on extending database technology (EDBT'15), pp 373–384

- Tian Y, Özcan F, Zou T, Goncalves R, Pirahesh H (2016) Building a hybrid warehouse: efficient joins between data stored in hdfs and enterprise warehouse. ACM Trans Database Syst 41(4):21:1–21:38
- Tomasic A, Raschid L, Valduriez P (1998) Scaling access to heterogeneous data sources with DISCO. IEEE Trans Know Data Eng (TKDE). 10(5):808–823
- Traverso M (2013) Presto: interacting with petabytes of data at facebook. <https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920>
- Vertica (2016) Hadoop integration guide – HP Vertica analytic database. https://my.vertica.com/docs/7.0.x/PDF/HP_Vertica_7.0.x_HadoopIntegration.pdf

Hybrid Transactional and Analytical Processing

- ▶ Hybrid OLTP and OLAP
-

Hybrid Warehouse

- ▶ Hybrid Systems Based on Traditional Database Extensions



Impala

Marcel Kornacker¹ and Alex Behm²

¹Blink Computing, San Francisco, CA, USA

²Databricks, San Francisco, CA, USA

Overview

Apache Impala is a modern MPP SQL engine architected specifically for the Hadoop data processing environment. Impala provides low latency and high concurrency for business intelligence/analytic workloads, which are not served well by batch-oriented processing frameworks such as MapReduce or Spark. Unlike traditional commercial analytic DBMSs, Impala is not a monolithic system but instead relies on the functionality of common, shared components of the Hadoop ecosystem to provide RDBMS-like functionality. In particular, Impala utilizes Hive's Metastore as a system catalog, which it shares with other processing frameworks such as Hive and Spark. In addition, Impala does not provide persistence functionality itself; instead, it relies on storage management components of the Hadoop ecosystem (e.g., HDFS, Kudu), which it drives via their public APIs. These storage management components in effect implement the record service layer of a traditional RDBMS and define the degree to which Impala can support traditional update capabilities and transactional semantics. As an example, HDFS

provides append-only file system functionality, which Impala can take advantage of to provide bulk insert functionality. However, single-row insertions have poor performance, and updates are not supported at all. In contrast, an online record-oriented storage manager such as Kudu supports both single-row inserts and updates (and deletes), which allows Impala to support the standard SQL functionality for inserts and updates.

This chapter presents Impala from a user's perspective and gives an overview and motivation of its architecture and main components.

Introduction

Impala is an open-source, state-of-the-art MPP SQL query engine designed specifically to leverage the flexibility and scalability of the Hadoop ecosystem. Impala's goal is to provide the familiar SQL support and multiuser performance of a traditional analytic DBMS, but to do so in an environment that also supports multiple, non-SQL data processing frameworks (e.g., MapReduce (Dean and Ghemawat 2004), Spark (Zahari et al. 2010)) and to that end utilizes standardized, open file formats and data access APIs. Impala's development started in 2011 and its initial version was released in May of 2013. Impala has been an open-source project since its beta release, and it became a top-level Apache project in November of 2017.

Unlike other popular MPP analytic DBMSs, several of which are forks of Postgres, Impala was written from scratch in C++ and Java. It maintains Hadoop's flexibility with regard to the choice of processing frameworks by utilizing standard components (HDFS <http://hadoop.apache.org/>, Kudu <https://kudu.apache.org/>, Hive Metastore <https://hive.apache.org/>, Sentry <https://sentry.apache.org/>, etc.) and is able to read the majority of the file formats that are popular with Hadoop users (e.g., text, Parquet <https://parquet.apache.org/>, Avro <https://avro.apache.org/>, etc.). Impala implements principles pioneered for traditional parallel DBMSs, such as data-partitioned parallelism and pipelining, and runs on a collection of daemon processes that are responsible for all aspects of query execution and that run on the same machines as the rest of the Hadoop infrastructure.

User View of Impala

Impala is a query engine which is integrated into the Hadoop environment and relies on a number of standard Hadoop infrastructure components to deliver an RDBMS-like experience. The standardized file formats and component APIs that are part of Hadoop allow Impala to offer very broad SQL analytic functionality against a wide variety of data formats as well as storage options. The latter include on-premises storage as well as cloud object storage such as AWS S3 and Azure Data Lake Store.

However, due to the looser coupling between the query engine and the storage managers, there are some notable differences in the areas of supported SQL update functionality and transactional semantics. In both cases, Impala surfaces the functionality made available by the storage managers themselves but does not try to augment it to bring it up to a uniform functional standard.

Another aspect in which Impala differs from a monolithic analytic DBMS is the approach to physical schema design. Physical schema design allows the user to optimize the physical layout of the table data to match the require-

ments of the analytic workload and is therefore an important aspect of performance optimization. Since Impala does not manage physical data itself, the functionality related to physical schema design that is surfaced in Impala also depends on what is supported by the storage managers.

Impala was targeted at standard business intelligence environments and to that end supports most relevant industry standards: clients connect via ODBC or JDBC, authentication is handled via Kerberos or LDAP, and authorization functionality (supplied via Sentry) follows the standard SQL roles and privileges.

In order to query data that resides in one of the supported Hadoop storage managers, the user creates tables via the familiar CREATE TABLE statement. This provides the logical schema of the data as well as the storage manager-specific parameters needed for Impala to access the underlying physical data. Subsequently the user can then reference that table in a SELECT statement, with the familiar relational semantics and operations (joins, grouping, etc.). In particular, a single SELECT statement can reference multiple tables, regardless of their underlying storage managers, making it possible to join on-premises HDFS data with S3-resident data, etc.

Complex Types Support

Data ingested into Hadoop often arrives in a non-relational form because the source system is not relational or because the data is hierarchical and more naturally produced in a de-normalized fashion. Preserving the original structure of the data eliminates the need to flatten and remodel the data before querying it and alleviates the cognitive burden for users who are used to the existing structure. To this end, Impala supports querying tables with the complex types *array*, *map*, and *struct*, which are familiar from Hive and supported in the shared Hive metastore. At present, this functionality is only available for the Parquet file format but in principle can be supported for any nested relational file format, including JSON, Avro, etc. Parquet stores nested data using a

Dremel-style shredding (Melnik et al. 2010) that enables the full benefits of a columnar format for arbitrarily nested data.

Impala enhances its SQL syntax to query complex types in a concise and user-friendly way. The new syntax was designed to be a natural extension of SQL and provide the full functionality of SQL on complex types. The main ideas are summarized as follows:

- Nested fields are accessed with a familiar “dot” notation, e.g., `tbl.col.nested1.nested2`.
- The collection types *map* and *array* must be referenced in the `FROM` clause just like conventional tables. This flattens the nested collection via an implicit parent-child join and exposes the nested fields as columns that are accessible as usual via dot-path expressions.
- Collections can be referenced through absolute paths (`FROM tbl.nested_collection`) or relative paths (`FROM tbl t, t.nested_collection`). The former exposes the fields in “`nested_collection`” but not those in the parent “`tbl`,” while the latter exposes both the parent and child. Subqueries (including inline views) may use correlated relative paths in the `FROM` clause that reference maps/arrays from enclosing blocks. This allows a SQL-in-SQL pattern where the data in nested collections can be filtered/joined/aggregated with the full expressiveness of SQL.

The following query from (Behm 2015) illustrates the syntax extensions. It lists all customers that have at least five orders that were placed on a date where the customer also had at least one support call and web interaction.

```
SELECT cid, name, cnt
FROM customers c,
     (SELECT COUNT(oid) cnt
      FROM c.orders o
     WHERE order_date IN (SELECT visit_date
                           FROM c.web_visits)
       AND order_date IN (SELECT call_date
                           FROM c.support_calls))
HAVING cnt >= 5) v
```

Impala + HDFS

HDFS is the most widely used system for storing data in Hadoop and offers the highest performance for read-oriented workloads. It provides a file system API with append-only semantics and gives the user a choice of multiple file formats. As a consequence, for HDFS-resident data, Impala supports the SQL `SELECT` and `INSERT` statements; the latter writes at least one new file per statement and is therefore only recommended for bulk data insertion. The SQL `UPDATE` and `DELETE` statements are not supported.

When creating a table in HDFS, the user specifies the root storage location of the physical data as well as the file format, and possibly a list of partition columns, as in:

```
CREATE TABLE T (...)
PARTITIONED BY (day int, month int)
LOCATION '<hdfs-path>' STORED AS
    PARQUET;
```

For an unpartitioned table, data files are stored by default directly in the named storage location. For a partitioned table, data files are placed in subdirectories which themselves are located in the named storage location and whose paths embed the partition columns’ values. As an example, for `day = 17, month = 2` of Table T, all data files would be located in `<hdfs-path>/day = 17/month = 2/`. This form of partitioning separates the table data physically and allows static and dynamic partition pruning at runtime; however, it does not physically cluster the data files belonging to an individual partition (the blocks of those data files are spread across all HDFS DataNodes).

The file format should be chosen with an eye toward the provenance of the data and target application. Impala supports most popular file formats, including Parquet, text, Avro, RC, and sequence files, in combination with a number of compression algorithms (snappy, gzip, bz2, etc.). As a rule of thumb, row-oriented formats (e.g., text, Avro) favor small incremental data loads, whereas column-oriented formats (e.g., Parquet) provide superior performance for analytic queries.

Impala + Kudu

Kudu has a record-oriented interface capable of single-record inserts, updates, and lookups. In addition to that, it also supports sophisticated options for physical data partitioning: whereas HDFS only supports value-based partitioning, Kudu supports hash and range partitioning, which gives the user more flexibility in matching the physical data layout to the requirements of the workload. Range partitions allow partition pruning against a broad class of predicates (much like value-based partitioning, except that the number of partitions can be controlled more easily), and hash partitions facilitate value-based insertions (but otherwise only allow partition pruning for equality predicates).

To continue the previous example:

```
CREATE TABLE T (... , PRIMARY KEY (id,
    date))
PARTITION BY HASH (id) PARTITIONS 4
RANGE (date) (PARTITION VALUES ...)
STORED AS KUDU;
```

In this example, the user creates a table with partitions that reflect the hash value of the id column as well as the declared value range of the date column.

Architecture

Impala is a massively parallel SQL execution engine which is designed to run on hundreds of machines in existing Hadoop clusters. Its approach to parallelization and the way this is implemented follow the techniques pioneered for parallel database systems. Figure 1 gives an overview.

An Impala deployment is comprised of the following components:

- **Statestored:** The statestore daemon is a generic publish/subscribe service that disseminates shared metadata throughout the cluster. There is a single statestored instance for the entire cluster.
- **Catalogd:** The catalog daemon collects and synchronizes table metadata with other

Hadoop ecosystem components (such as the Hive metastore, the HDFS NameNode, and the Sentry Service). The catalog daemon uses the statestore to distribute a unified view of table metadata to the Impala daemons. There is a single catalogd instance for the entire cluster.

- **Impalad:** Impala daemons accept client requests and are responsible for query planning, coordination, and execution. They cache metadata received from the catalogd and are the only components actively involved in query execution. There is typically one impalad running on each node in the cluster.

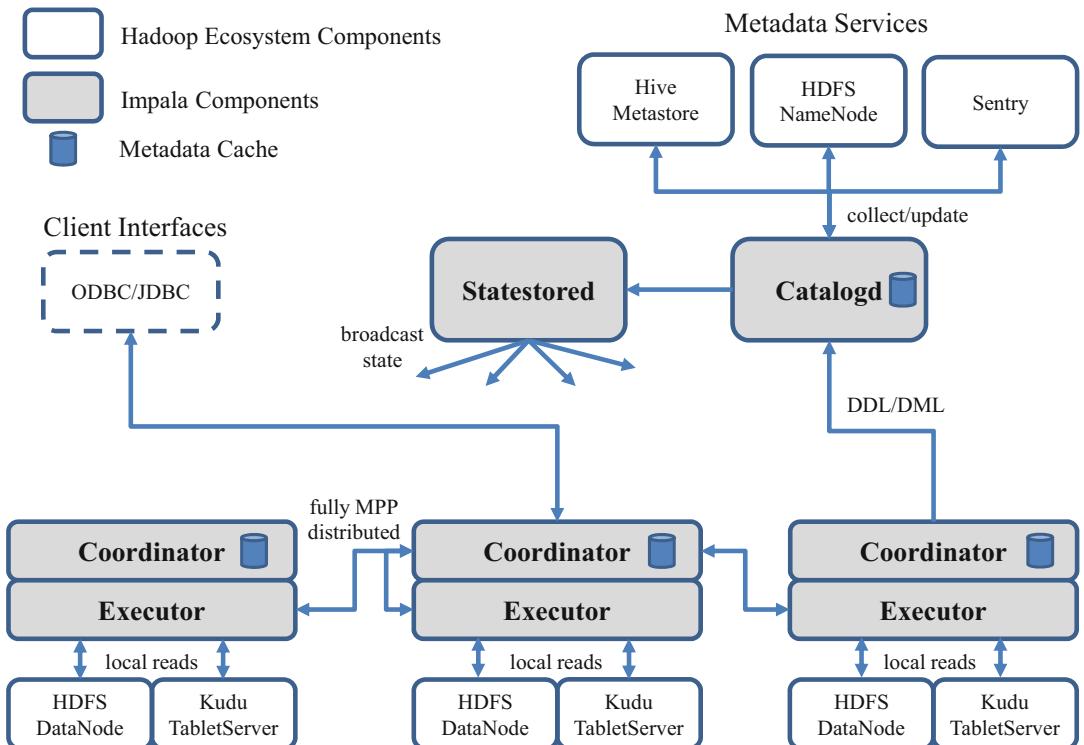
The following sections describe each daemon in more detail.

Statestore Daemon

The statestore is used to disseminate table metadata originating from the catalog daemon as well as membership and load information for the entire cluster. Its information is organized in topics which are versioned collections of topic items. Each impalad registers itself with the statestore at startup at which point it receives an initial bulk update to synchronize it with the current state of all topics. Subsequent updates utilize the topic version numbers to send deltas; thereby avoiding having to resend entire topics whenever a single piece of metadata changes.

The statestore also sends periodic keepalive messages to all registered impalads in order to detect node failures and update its membership information. Membership itself also has a topic, and changes are immediately disseminated using the generic topic update mechanism.

Given the asynchronous nature of how topic changes are broadcast, it is possible for two impalads to be caching different versions of any topic, e.g., the table metadata. This inconsistency is not a problem during query processing, because the query coordinator creates an execution plan that embeds all relevant metadata. An executing impalad then references that metadata rather than its local cache.



Impala, Fig. 1 Overview of Impala’s architecture and Hadoop ecosystem components. Executors can also read from cloud blob stores like S3 and ADLS (not shown)

Catalog Daemon

Impala’s catalogd has two purposes: (a) it collects and disseminates catalog data to impalads via the statestore broadcast mechanism, and (b) it serves as the liaison between the impalads and the external persistent metadata repository, the Hive metastore (HMS). The catalog daemon allows Impala daemons to execute DDL that is persisted in the HMS, and it synchronizes its view of the cluster’s metadata from the HMS, thereby allowing interoperability between Impala and other Hadoop processing frameworks that rely on the HMS for metadata storage.

The catalog service pulls metadata that is relevant for query planning and execution from the appropriate components of the Hadoop ecosystem: the Hive metastore for logical table definitions; the HDFS namenode for information about data files, blocks; and the Sentry Service for authorization metadata such as roles and privileges. That metadata is aggregated into a unified catalog structure. Any changes to that catalog (e.g., when

a new table is created) are disseminated via the statestore and cached by the Impala daemons.

Since catalogs are often very large, and access to tables is rarely uniform, the catalog service only loads a skeleton entry for each table it discovers at startup. The full table metadata can be loaded lazily in the background or on request, when it is first referenced. If an incoming query references a table before its full metadata has been loaded, the impalad acting as the query coordinator issues a synchronous prioritization request to the catalog service, which blocks the query until the query-relevant metadata has been loaded.

Impala Daemon

This process assumes one or both of these two roles:

- A query coordinator, which accepts client requests (submitted via Thrift), generates an ex-

ecution plan, and then coordinates the execution across the participating Impala executors in the cluster

- A query executor, which accepts execution plan fragments from a coordinator and executes them

There is no central “master” process that handles query coordination for all client requests, and the actual roles assumed by each deployed impalad process are configurable so that it is possible to independently scale up the number of coordinators and executors according to the throughput requirements of the workload.

In a typical on-premise deployment of Impala, one impalad process runs on every node that also runs a DataNode process (for HDFS) or tablet server process (for Kudu). This allows Impala to take advantage of data locality and do a maximal amount of local preprocessing before sending data across the network.

One design goal for Impala is to avoid synchronous RPCs to shared cluster components during query execution, which would severely limit throughput scalability for concurrent workloads. For that reason, each impalad coordinator caches all of the metadata relevant for query planning.

Query Planning

Impalads with the coordinator role are responsible for compiling SQL text into query plans executable by the Impala executors. Impala’s query compiler is written in Java and consists of a fully featured SQL parser and cost-based query optimizer. In addition to the basic SQL features (select, project, join, group by, order by, limit), Impala supports in-line views, uncorrelated and correlated subqueries (transformed into joins), and all variants of outer joins as well as explicit left/right semi- and anti-joins, analytic window functions, and extensions for nested data.

The query compilation process follows a traditional division into stages: query parsing, semantic analysis, and query planning/optimization. The latter stage constructs an executable

plan in two phases: (1) single-node planning and (2) plan parallelization and fragmentation.

In the first phase, the parse tree is translated into a single-node plan tree consisting of the following operators: scan (HDFS, Kudu, HBase), hash join, cross join, union, hash aggregation, sort, top-n, analytic evaluation, and subplan (for nested data evaluation). This phase is responsible for expression rewrites (e.g., constant folding), converting subqueries into joins, assigning predicates at the lowest possible operator in the tree, inferring predicates based on equivalence classes, pruning table partitions, setting limits/offsets, and applying column projections, as well as performing cost-based optimizations such as join ordering and coalescing analytic window functions to minimize total execution cost. Cost estimation is based on table and partition cardinalities plus distinct value counts for each column. Impala uses a quadratic algorithm to determine join order, instead of the more common dynamic programming approach, due to the absence of “interesting orders” (all operators are hash based).

The second planning phase takes the single-node plan as input and produces a distributed execution plan, consisting of a tree of plan fragments. A plan fragment is the basic unit of work sent to executors. It consumes data from any number of child fragments and sends data to at most one parent fragment. The general goal is to minimize data movement and maximize scan locality (in HDFS, local reads are considerably cheaper than remote ones). This is done by adding exchange operators to the plan where necessary and by adding extra operators to minimize data movement across the network (e.g., local pre-aggregation). This second optimization phase decides the join strategy (broadcast or partitioning) for every join operator (the join order is fixed at that point). A broadcast join replicates the entire build side of a join to all machines executing the hash join; a partitioning join repartitions both join inputs on the join expressions. Impala takes into account the existing data partitions of the join inputs and chooses the strategy that is estimated to minimize the amount of data transferred over the network.

Query Execution

An impalad with the executor role receives plan fragments from the coordinator and is responsible for their execution. It is written in C++ and was designed specifically to take advantage of modern hardware, among other things, by utilizing runtime code generation. The latter reduces the instruction count and memory overhead at runtime when compared to a more traditional, interpretive approach and to other engines implemented in Java.

Impala's execution model follows that of traditional parallel databases: it implements Volcano-style (Graefe 1990) distributed execution with Exchange operators and batch-at-a-time processing. With the exception of "stop-and-go" operators (e.g., sort), the execution is fully pipelined, which minimizes memory consumption for intermediate results. Impala employs a canonical row-oriented in-memory format which is separate from how data is represented on disk; this is necessary in order to allow Impala to support a range of file formats. Operators that may need to consume lots of memory are designed to be able to spill parts of their working set to external storage if needed. Spillable operators are hash join, hash aggregation, sort, and analytic function evaluation.

Runtime code generation using LLVM (Lattner and Adve 2004) is at the core of Impala's techniques for execution efficiency. LLVM allows applications like Impala to perform just-in-time compilation within a running process, with the full benefits of a modern optimizer and the ability to generate machine code for a number of architectures. Impala utilizes this to produce query-specific versions of functions that are critical to performance, namely, those "inner-loop" functions that are executed many times in a given query (e.g., for every tuple), and thus constitute the majority of the cycles executed on behalf of the query.

When running queries against HDFS-resident data, Impala takes advantage of two HDFS features to retrieve data at or near hardware speed: (a) short-circuit local reads (HDFS short-circuit local reads n.d.) allow the client to bypass

HDFS's DataNode protocol and read directly from local disk instead, in the case where the reader and the data are collocated and (b) in-memory caching (Centralized cache management n.d.) allows the client to map cached data directly into its address space, thereby avoiding extra data copies and checksums.

Further, Impala implements many of the advanced performance features pioneered by parallel RDBMS such as semi-join reductions with Bloom filters, dynamic partition pruning, and runtime data elimination by evaluating predicates against min/max column values and dictionaries inside data files.

Cross-References

- ▶ [Hive](#)
- ▶ [Spark SQL](#)

References

- Behm A (2015) New in Cloudera Enterprise 5.5: support for complex types in Impala. <https://blog.cloudera.com/blog/2015/11/new-in-cloudera-enterprise-5-5-support-for-complex-types-in-impala/>
- Centralized cache management in HDFS. <https://hadoop.apache.org/docs/r2.3.0/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html>
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th symposium on operating systems design and implementation
- Graefe G (1990) Encapsulation of parallelism in the Volcano query processing system. In: Proceedings of the 1990 ACM SIGMOD international conference on management of data
- HDFS short-circuit local reads. <http://hadoop.apache.org/docs/r2.5.1/hadoop-projectdist/hadoop-hdfs/ShortCircuitLocalReads.html>
- Lattner C, Adve V (2004) LLVM: a compilation framework for lifelong program analysis & transformation. In: Proceedings of the international symposium on code generation and optimization, 2004
- Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T (2010) Dremel: interactive analysis of web-scale datasets. Proc VLDB Endow 3(1):330–339
- Zahari M et al (2010) Spark: cluster computing with working sets. In: 2nd USENIX workshop on hot topics in cloud computing

Incorruptible Big Data

► Data Replication and Encoding

Incremental Approximate Computing

Do Le Quoc¹, Dhanya R Krishnan¹,
Pramod Bhatotia², Christof Fetzer¹, and Rodrigo Rodrigues³

¹TU Dresden, Dresden, Germany

²The University of Edinburgh and Alan Turing Institute, Edinburgh, UK

³INESC-ID, IST (University of Lisbon), Lisbon, Portugal

Introduction

Stream analytics systems are an integral part of modern online services. These systems are extensively used for transforming raw data into useful information. Much of this raw data arrives as a continuous data stream and in huge volumes, requiring real-time stream processing based on parallel and distributed computing frameworks (Bhatotia et al. 2014; Murray et al. 2013; Zaharia et al. 2013).

Near real-time processing of data streams has two desirable, but contradictory, design requirements (Murray et al. 2013; Zaharia et al. 2013): (i) achieving low latency and (ii) efficient resource utilization. For instance, the low-latency requirement can be met by employing more computing resources and parallelizing the application logic over the distributed infrastructure. Since most data analytics frameworks are based on the data-parallel programming model (Dean and Ghemawat 2004), almost linear scalability can be achieved with increased computing resources (Quoc et al. 2013, 2014, 2015a,b). However, low latency comes at the cost of lower throughput and ineffective utilization of the computing resources. Moreover, in some cases, processing all data items of the input stream would require

more than the available computing resources to meet the desired SLAs or the latency guarantees.

To strike a balance between these two contradictory goals, there is a surge of new computing paradigms that prefer to compute over a subset of data items instead of the entire data stream. Since computing over a subset of the input requires less time and resources, these computing paradigms can achieve bounded latency and efficient resource utilization. In particular, two such paradigms are incremental and approximate computing.

Incremental computing. Incremental computation is based on the observation that many data analytics jobs operate incrementally by repeatedly invoking the same application logic or algorithm over an input data that differs slightly from that of the previous invocation (Bhatotia et al. 2015; Gunda et al. 2010; He et al. 2010). In such a workflow, small, localized changes to the input often require only small updates to the output, creating an opportunity to update the output incrementally instead of recomputing everything from scratch (Acar 2005; Bhatotia 2015). Since the work done is often proportional to the change size rather than the total input size, incremental computation can achieve significant performance gains (low latency) and efficient utilization of computing resources (Bhatotia et al. 2011b, 2012b; Popa et al. 2009).

The most common way for incremental computation is to rely on programmers to design and implement an application-specific incremental update mechanism (or a *dynamic algorithm*) for updating the output as the input changes (Brodal and Jacob 2002; Chiang and Tamassia 1992). While dynamic algorithms can be asymptotically more efficient than recomputing everything from scratch, research in the algorithms community shows that these algorithms can be difficult to design, implement, and maintain even for simple problems. Furthermore, these algorithms are studied mostly in the context of the uniprocessor computing model, making them ill-suited for parallel and distributed settings which are commonly used for large-scale data analytics.

Recent advancements in self-adjusting computation (Acar 2005; Acar et al. 2010; Ley-Wild et al. 2009) overcome the limitations of dynamic algorithms. Self-adjusting computation transparently benefits existing applications, without requiring the design and implementation of dynamic algorithms. At a high level, self-adjusting computation enables incremental updates by creating a dynamic dependence graph of the underlying computation, which records control and data dependencies between the sub-computations. Given a set of input changes, self-adjusting computation performs change propagation, where it reuses the *memoized* intermediate results for all sub-computations that are unaffected by the input changes, and recomputes only those parts of the computation that are transitively affected by the input change. As a result, self-adjusting computation computes only on a subset (*delta*) of the computation instead of recomputing everything from scratch.

Approximate computing. Approximate computation is based on the observation that many data analytics jobs are amenable to an approximate, rather than the exact, output (Krishnan et al. 2016; Quoc et al. 2017b,d). For such an approximate workflow, it is possible to trade accuracy by computing over a partial subset instead of the entire input data to achieve low latency and efficient utilization of resources.

Over the last two decades, researchers in the database community proposed many techniques for approximate computing including sampling (Al-Kateb and Lee 2010), sketches (Cor-mode et al. 2012), and online aggregation (Heller-stein et al. 1997). These techniques make different trade-offs with respect to the output quality, supported query interface, and workload. However, the early work in approximate computing was mainly targeted toward the centralized database architecture, and it was unclear whether these techniques could be extended in the context of big data analytics.

Recently, sampling-based approaches have been successfully adopted for distributed data analytics (Agarwal et al. 2013; Goiri et al. 2015). These systems show that it is possible to have a

trade-off between the output accuracy and performance gains (also efficient resource utilization) by employing sampling-based approaches for computing over a *subset* of data items. However, these “big data” systems target batch processing workflow and cannot provide required low-latency guarantees for stream analytics.

Combining incremental and approximate computing. This entry makes the observation that the two computing paradigms, incremental and approximate computing, are complementary. Both computing paradigms rely on computing over a subset of data items instead of the entire dataset to achieve low latency and efficient cluster utilization. Therefore, this entry proposes to combine these paradigms together in order to leverage the benefits of both. Furthermore, incremental updates can be achieved without requiring the design and implementation of application-specific dynamic algorithms and support approximate computing for stream analytics.

The high-level idea is to design a sampling algorithm that biases the sample selection to the memoized data items from previous runs. This idea is realized by designing an online sampling algorithm that selects a representative subset of data items from the input data stream. Thereafter, the sample is biased to include data items for which already there are memoized results from previous runs while preserving the proportional allocation of data items of different (strata) distributions. Next, the user-specified streaming query is performed on this biased sample by making use of self-adjusting computation, and the user is provided with an incrementally updated approximate output with error bounds.

INCAPPROX

System Overview

INCAPPROX is designed for real-time data analytics on online data streams. At the high level, the online data stream consists of data items from diverse sources of events or sub-streams. INCAPPROX uses a stream aggregator

(e.g., Apache Kafka ([Kafka – A high-throughput distributed messaging system](#))) that integrates data from these sub-streams, and thereafter, the system reads this integrated data stream as the input.

INCAPPROX facilitates user querying on this data stream and supports a user interface that consists of a streaming query and a query budget. The user submits the streaming query to the system as well as specifies a query budget. The query budget can either be in the form of latency guarantees/SLAs for data processing, desired result accuracy, or computing resources available for query processing. INCAPPROX makes sure that the computation done over the data remains within the specified budget. To achieve this, the system makes use of a mixture of incremental and approximate computing for real-time processing over the input data stream and emits the query result along with the confidence interval or error bounds.

System Model

In this section, the system model assumed in this work is presented.

Programming model. INCAPPROX supports a *batched stream processing* programming model. In batched stream processing, the online data stream is divided into small batches or sets of records; and for each batch, a distributed data-parallel job is launched to produce the output.

Computation model. The computation model of INCAPPROX for stream processing is *sliding window computations*. In this model the computation window slides over the input data stream, where new arriving input data items are added to the window, and the old data items are dropped from the window as they become less relevant to the analysis.

In sliding window computations, there is a substantial overlap of data items between the two successive computation windows, especially, when the size of the window is large relative to the slide interval. This overlap of unchanged data items provides an opportunity to update the output incrementally.

Assumptions. INCAPPROX makes the following assumptions. Possible methods to enforce them are discussed in section “[Discussion](#)” and in the entry Krishnan et al. (2016).

1. This work assumes that the input stream is stratified based on the source of event, i.e., the data items within each stratum follow the same distribution and are mutually independent. Here a *stratum* refers to one sub-stream. If multiple sub-streams have the same distribution, they are combined to form a stratum.
2. This work assumes the existence of a virtual function that takes the user-specified budget as the input and outputs the sample size for each window based on the budget.

Building Blocks

Techniques and the motivation behind INCAPPROX’s design choices are briefly described.

Stratified sampling. In a streaming environment, since the window size might be very large, for a realistic rate of execution, INCAPPROX performs approximation using samples taken within the window. But the data stream might consist of data from disparate events. As such, INCAPPROX needs to make sure that every sub-stream is considered fairly to have a representative sample from each sub-stream. For this the system uses stratified sampling (Al-Kateb and Lee 2010). Stratified sampling ensures that data from every stratum is selected and none of the minorities are excluded. For statistical precision, INCAPPROX uses proportional allocation of each sub-stream to the sample (Al-Kateb and Lee 2010). It ensures that the sample size of each sub-stream is in proportion to the size of sub-stream in the whole window.

Self-adjusting computation. For incremental sliding window computations, INCAPPROX uses self-adjusting computation (Acar 2005; Acar et al. 2010) to reuse the intermediate results of sub-computations across successive runs of jobs. In this technique the system maintains a dependence graph between sub-computations

of a job and reuses memoized results for sub-computations that are unaffected by the changed input in the computation window.

Algorithm

This section presents an overview algorithm of INCAPPROX. The algorithm computes a user-specified streaming *query* as a sliding window computation over the input data stream. The user also specifies a query *budget* for executing the query, which is used to derive the sample size (*sampleSize*) for the window using a cost function (see sections “Assumptions” and “Discussion”). The cost function ensures that processing remains within the query budget.

For each window, INCAPPROX first adjusts the computation window to the current start time t by removing all old data items from the *window* (*timestamp* $< t$). Similarly, the system also drops all old data items from the list of memoized items (*memo*) and the respective memoized results of all sub-computations that are dependent on those old data items.

Next, the system reads the new incoming data items in the *window*. Thereafter, it performs proportional stratified sampling (detailed in Krishnan et al. 2016) on the *window* to select a sample of size provided by the cost function. The stratified sampling algorithm ensures that samples from all strata are proportional and no stratum is neglected.

Next, INCAPPROX biases the stratified sample to include items from the memoized sample, in order to enable the reuse of memoized results from previous sub-computations. The biased sampling algorithm biases samples *specific to each stratum*, to ensure reuse and, at the same time, retain proportional allocation (Krishnan et al. 2016).

Thereafter, on this biased sample, the system runs the user-specified *query* as a data-parallel job *incrementally*, i.e., it reuses the memoized results for all data items that are unchanged in the window, and updates the output based on the changed (or new) data items. After the job finishes, the system memoizes all the items in the sample and their respective sub-computation

results for reuse for the subsequent windows (Krishnan et al. 2016).

The job provides an estimated output which is bound to a range of error due to approximation. INCAPPROX performs error estimation to estimate this error bound and define a confidence interval for the result as $output \pm error\ bound$ (Krishnan et al. 2016).

The entire process repeats for the next window, with updated windowing parameters and the sample size. (Note that the query budget can be updated across windows during the course of stream processing to adapt to the user’s requirements.)

Estimation of Error Bounds

In order to provide a confidence interval for the approximate output, the error bounds are computed due to approximation.

Approximation for aggregate functions. Aggregate functions require results based on all the data items or groups of data items in the population. But since the system computes only over a small sample from the population, an *estimated* result based on the weightage of the sample can be obtained.

Consider an input stream S , within a window, consisting of n disjoint strata $S_1, S_2 \dots, S_n$, i.e., $S = \sum_{i=1}^n S_i$. Suppose the i th stratum S_i has B_i items and each item j has an associated value v_{ij} . Consider an example of taking sum of these values across the whole window, represented as $\sum_{i=1}^n (\sum_{j=1}^{B_i} v_{ij})$. To find an approximate sum, INCAPPROX first selects a sample from the window based on stratified and biased sampling, i.e., from each i th stratum S_i in the window, the system samples b_i items. Then it estimates the sum from this sample as $\hat{\tau} = \sum_{i=1}^n (\frac{B_i}{b_i} \sum_{j=1}^{b_i} v_{ij}) \pm \epsilon$ where the error bound ϵ is defined as:

$$\epsilon = t_{f,1-\frac{\alpha}{2}} \sqrt{\widehat{Var}(\hat{\tau})} \quad (1)$$

Here, $t_{f,1-\frac{\alpha}{2}}$ is the value of the t-distribution (i.e., *t-score*) with f degrees of freedom and $\alpha = 1 - \text{confidence level}$. The degree of freedom f is expressed as:

$$f = \sum_{i=1}^n b_i - n \quad (2)$$

The estimated variance for sum, $\widehat{Var}(\hat{\tau})$ is represented as:

$$\widehat{Var}(\hat{\tau}) = \sum_{i=1}^n B_i * (B_i - b_i) \frac{s_i^2}{b_i} \quad (3)$$

where s_i^2 is the population variance in the i th stratum. Since the bias sampling is such that the statistics of stratified sampling is preserved, the statistical theories (Lohr 2009) for stratified sampling has been used to compute the error bound.

Currently, INCAPPROX supports error estimation only for aggregate queries. For supporting queries that compute extreme values, such as minimum and maximum, the system can make use of extreme value theory (Coles 2001; Liu and Meeker 2014) to compute the error bounds.

Error bound estimation. For error bound estimation, first the sample statistic used to estimate a population parameter is identified, e.g., *sum*, and a desired confidence level is selected, e.g., 95%. In order to compute the margin of error ϵ using t-score as given in Eq. 1, the sampling distribution must be nearly normal. The central limit theorem (CLT) states that when the size of sample is sufficiently large ($>= 30$), then the sampling distribution of a statistic approximates to *normal distribution*, regardless of the underlying distribution of values in the data (Lohr 2009). Hence, INCAPPROX computes t-score using a t-distribution calculator ([The Apache Commons Mathematics Library](#)), with the given degree of freedom f (see Eq. 2), and cumulative probability as $1 - \alpha/2$ where $\alpha = 1 - \text{confidence level}$ (Lohr 2009). Thereafter, the system estimates the variance using the corresponding equation for the sample statistic considered (for *sum*, the Equation is (3)). Finally, the system uses this t-score and estimated variance of the sample statistic and computes the margin of error using Eq. 1.

Discussion

The design of INCAPPROX is based on the assumptions in section “[Assumptions](#)”. Solving these assumptions is beyond the scope of this entry; however, this section discusses some of the approaches that could be used to meet the assumptions.

I: Stratification of sub-streams. Currently this work assumes that sub-streams are stratified, i.e., the data items of individual sub-streams have the same distribution. However, it may not be the case. Next, two alternative approaches are discussed, namely, bootstrap (Dziuda 2010) and a semi-supervised learning algorithm (Masud et al. 2012) to classify evolving data streams. Bootstrap (Dziuda 2010; Efron and Tibshirani 1986) is a nonparametric resampling technique used to estimate parameters of a population. It works by randomly selecting a large number of bootstrap samples with replacement and with the same size as in the original sample. Unknown parameters of a population can be estimated by averaging these bootstrap samples. Such a bootstrap-based classifier could be created from the initial reservoir of data, and the classifier could be used to classify sub-streams. Alternatively, a semi-supervised algorithm (Masud et al. 2012) can be employed to stratify a data stream. This algorithm manipulates both unlabeled and labeled data items to train a classification model.

II: Virtual cost function. Secondly, this work assumes that there exists a virtual function that computes the sample size based on the user-specified query budget. The query budget could be specified as either available computing resources or latency requirements. Two existing approaches – Pulsar (Angel et al. 2014) and resource prediction model (Wieder et al. 2010a,b, 2012; Ganapathi 2009) – can be used to design such a virtual function for the given computing resources and latency requirements, respectively (see details in Krishnan et al. 2016).

Related Work

INCAPPROX builds on two computing paradigms, namely, incremental and approximate computing. This section provides a survey of techniques proposed in these two paradigms.

Incremental computation. To apply incremental computing, the earlier big data systems (Google’s Percolator (Peng and Dabek 2010) and Yahoo’s CBP (Logothetis et al. 2010) proposed an alternative programming model where the programmer is asked to implement an efficient incremental-update mechanism. However, these systems depart from the existing programming model, and also require implementation of dynamic algorithms on per-application basis, which could be difficult to design and implement.

To overcome the limitations, researchers proposed transparent approaches for incremental computation. Examples of transparent systems include Incoop (Bhatotia et al. 2011a,b), DryadInc (Popa et al. 2009), Slider (Bhatotia et al. 2012a, 2014), and NOVA (Olston et al. 2011). These systems leverage the underlying data-parallel programming model such as MapReduce (Dean and Ghemawat 2004) or Dryad (Isard et al. 2007) for supporting incremental computation. INCAPPROX was built on transparent big data systems for incremental computation. In particular, the system leverages the advancements in self-adjusting computation (Acar 2005; Bhatotia 2015) to improve the efficiency of incremental computation. In contrast to the existing approaches, INCAPPROX extends incremental computation with the idea of approximation, thus further improving the performance.

Approximate computation. Approximation techniques such as sampling (Al-Kateb and Lee 2010), sketches (Cormode et al. 2012), and online aggregation (Hellerstein et al. 1997) have been well-studied over the decades in the context of traditional (centralized) database systems. In the last 5 years, sampling-based techniques have

been successfully employed in many distributed data analytics systems (Agarwal et al. 2013; Goiri et al. 2015; Quoc et al. 2017a,b). The systems such as ApproxHadoop (Goiri et al. 2015) and BlinkDB (Agarwal et al. 2013) showed that it is possible to achieve the benefits of approximate computing also in the context of distributed big data analytics. However, these systems target batch processing and are not able to support low-latency stream analytics.

Recently, StreamApprox (Quoc et al. 2017c,d) proposed an online sampling algorithm to “on-the-fly” take samples of input data streams in a distributed manner. Interestingly, the sampling algorithm is generalizable to two prominent types of stream processing models: batched and pipelined stream processing models. Lastly, PrivApprox (Quoc et al. 2017a,b) employed a combination of randomized response and approximate computation to support privacy-preserving stream analytics.

INCAPPROX builds on the advancements in approximate computing for big data analytics. However, the system is different from the existing approximate computing systems in two crucial aspects. First, unlike the existing systems, ApproxHadoop and BlinkDB, which are designed for batch processing, INCAPPROX is targeted at stream processing. Second, the system benefits from both approximate and incremental computing.

Conclusion

This entry presents the combination of incremental and approximate computations. The proposed approach transparently benefits unmodified applications, i.e., programmers do not have to design and implement application-specific dynamic algorithms or sampling techniques. The approach is based on the observation that both computing paradigms rely on computing over a subset of data items instead of computing over the entire dataset. These two paradigms are combined by designing a sampling algorithm that biases the

sample selection to the memoized data items from previous runs.

References

- Acar UA (2005) Self-adjusting computation. PhD thesis, Carnegie Mellon University
- Acar UA, Cotter A, Hudson B, Türkoğlu D (2010) Dynamic well-spaced point sets. In: Proceedings of the 26th annual symposium on computational geometry (SoCG)
- Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) Blinkdb: queries with bounded errors and bounded response times on very large data. In: Proceedings of the ACM European conference on computer systems (EuroSys)
- Al-Kateb M, Lee BS (2010) Stratified reservoir sampling over heterogeneous data streams. In: Proceedings of the 22nd international conference on scientific and statistical database management (SSDBM)
- Angel S, Ballani H, Karagiannis T, O'Shea G, Thereska E (2014) End-to-end performance isolation through virtual datacenters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Bhatotia P (2015) Incremental parallel and distributed systems. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS)
- Bhatotia P, Wieder A, Akkus IE, Rodrigues R, Acar UA (2011a) Large-scale incremental data processing with change propagation. In: Proceedings of the conference on hot topics in cloud computing (HotCloud)
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquini R (2011b) Incoop: MapReduce for incremental computations. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Bhatotia P, Dischinger M, Rodrigues R, Acar UA (2012a) Slider: incremental sliding-window computations for large-scale data analysis. In: Technical Report: MPI-SWS-2012-004
- Bhatotia P, Rodrigues R, Verma A (2012b) Shredder: GPU-accelerated incremental storage and computation. In: Proceedings of USENIX conference on file and storage technologies (FAST)
- Bhatotia P, Acar UA, Junqueira FP, Rodrigues R (2014) Slider: incremental sliding window analytics. In: Proceedings of the 15th international middleware conference (Middleware)
- Bhatotia P, Fonseca P, Acar UA, Brandenburg B, Rodrigues R (2015) iThreads: a threading library for parallel incremental computation. In: Proceedings of the 20th international conference on architectural support for programming languages and operating systems (ASPLOS)
- Brodal GS, Jacob R (2002) Dynamic planar convex hull. In: Proceedings of the 43rd annual IEEE symposium on foundations of computer science (FOCS)
- Chiang YJ, Tamassia R (1992) Dynamic algorithms in computational geometry. In: Proceedings of the IEEE
- Coles S (2001) An introduction to statistical modeling of extreme values. Springer, London/New York
- Cormode G, Garofalakis M, Haas PJ, Jermaine C (2012) Synopses for massive data: samples, histograms, wavelets, sketches. Found Trends Databases 4(1-3): 1–294
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Dziuda DM (2010) Data mining for genomics and proteomics: analysis of gene and protein expression data. Wiley, Hoboken
- Efron B, Tibshirani R (1986) Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. Stat Sci 1(1):54–75
- Ganapathi AS (2009) Predicting and optimizing system utilization and performance via statistical machine learning. In: Technical Report No. UCB/EECS-2009-181
- Goiri I, Bianchini R, Nagarakatte S, Nguyen TD (2015) ApproxHadoop: bringing approximations to MapReduce frameworks. In: Proceedings of the twenty-sixth international conference on architectural support for programming languages and operating systems (ASPLOS)
- Gunda PK, Ravindranath L, Thekkath CA, Yu Y, Zhuang L (2010) Nectar: automatic management of data and computation in datacenters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- He B, Yang M, Guo Z, Chen R, Su B, Lin W, Zhou L (2010) Comet: batched stream processing for data intensive distributed computing. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Hellerstein JM, Haas PJ, Wang HJ (1997) Online aggregation. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of the ACM European conference on computer systems (EuroSys)
- Kafka – A high-throughput distributed messaging system. <http://kafka.apache.org>. Accessed Nov 2017
- Krishnan DR, Quoc DL, Bhatotia P, Fetzer C, Rodrigues R (2016) IncApprox: a data analytics system for incremental approximate computing. In: Proceedings of the 25th international conference on world wide web (WWW)
- Ley-Wild R, Acar UA, Fluet M (2009) A cost semantics for self-adjusting computation. In: Proceedings of the annual ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)
- Liu S, Meeker WQ (2014) Statistical methods for estimating the minimum thickness along a pipeline. Technometrics 57(2):164–179
- Logothetis D, Olston C, Reed B, Web K, Yocum K (2010) Stateful bulk processing for incremental analytics. In:

- Proceedings of the ACM symposium on cloud computing (SoCC)
- Lohr S (2009) Sampling: design and analysis, 2nd edn. Cengage Learning, Boston
- Masud MM, Woolam C, Gao J, Khan L, Han J, Hamlen KW, Oza NC (2012) Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowl Inf Syst* 33(1):213–244
- Murray DG, McSherry F, Isaacs R, Isard M, Barham P, Abadi M (2013) Naiad: a timely dataflow system. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles (SOSP)
- Olston C et al (2011) Nova: continuous pig/hadoop workflows. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Peng D, Dabek F (2010) Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Popa L, Budiu M, Yu Y, Isard M (2009) DryadInc: reusing work in large-scale computations. In: Proceedings of the conference on hot topics in cloud computing (HotCloud)
- Quoc DL, Martin A, Fetzer C (2013) Scalable and real-time deep packet inspection. In: Proceedings of the 2013 IEEE/ACM 6th international conference on utility and cloud computing (UCC)
- Quoc DL, Yazdanov L, Fetzer C (2014) Dolen: user-side multi-cloud application monitoring. In: International conference on future internet of things and cloud (FI-CLOUD)
- Quoc DL, D'Alessandro V, Park B, Romano L, Fetzer C (2015a) Scalable network traffic classification using distributed support vector machines. In: Proceedings of the 2015 IEEE 8th international conference on cloud computing (CLOUD)
- Quoc DL, Fetzer C, Felber P, Rivière É, Schiavoni V, Sutra P (2015b) Unicrawl: a practical geographically distributed web crawler. In: Proceedings of the 2015 IEEE 8th international conference on cloud computing (CLOUD)
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017a) Privacy preserving stream analytics: the marriage of randomized response and approximate computing. <https://arxiv.org/abs/1701.05403>
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017b) PrivApprox: privacy-preserving stream analytics. In: Proceedings of the 2017 USENIX conference on USENIX annual technical conference (USENIX ATC)
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017c) Approximate stream analytics in Apache Flink and Apache Spark streaming. CoRR, abs/1709.02946
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017d) StreamApprox: approximate computing for stream analytics. In: Proceedings of the international middleware conference (Middleware)
- The Apache Commons Mathematics Library. <http://commons.apache.org/proper/commons-math>. Accessed Nov 2017
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010a) Brief announcement: modelling MapReduce for optimal execution in the cloud. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing (PODC)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010b) Conductor: orchestrating the clouds. In: Proceedings of the 4th international workshop on large scale distributed systems and middleware (LADIS)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2012) Orchestrating the deployment of computations in the cloud with conductor. In: Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI)
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles (SOSP)

Incremental Learning

- ▶ [Online Machine Learning Algorithms over Data Streams](#)
- ▶ [Online Machine Learning in Big Data Streams: Overview](#)
- ▶ [Recommender Systems over Data Streams](#)
- ▶ [Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#)

Incremental Sliding Window Analytics

Pramod Bhatotia¹, Umut A. Acar², Flavio P. Junqueira³, and Rodrigo Rodrigues⁴

¹The University of Edinburgh and Alan Turing Institute, Edinburgh, UK

²CMU, Pittsburgh, PA, USA

³Dell EMC, Hopkinton, MA, USA

⁴INESC-ID, IST (University of Lisbon), Lisbon, Portugal

Introduction

There is a growing need to analyze large feeds of data that are continuously collected. Either due to the nature of the analysis or in order to bound the computational complexity of analyzing

a monotonically growing data set, this processing often resorts to a *sliding window* analysis. In this type of processing, the scope of the analysis is limited to a recent interval over the entire collected data, and, periodically, newly produced inputs are appended to the window and older inputs are discarded from it as they become less relevant to the analysis.

The basic approach to sliding window data processing is to recompute the analysis over the entire window whenever the window slides. Consequently, even old, unchanged data items that remain in the window are reprocessed, thus consuming unnecessary computational resources and limiting the timeliness of results.

We can improve on this using an incremental approach, which normally relies on the programmers of the data analysis to devise an incremental update mechanism (He et al. 2010; Logothetis et al. 2010; Peng and Dabek 2010), i.e., an *incremental algorithm* (or *dynamic algorithm*) containing the logic for incrementally updating the output as the input changes. Research in the algorithms and programming languages communities shows that while such incremental algorithms can be very efficient, they can be difficult to design, analyze, and implement even for otherwise simple problems (Chiang and Tamassia 1992; Ramalingam and Reps 1993; Demetrescu et al. 2004; Acar 2005). Moreover, such incremental algorithms often assume a uniprocessor computing model and due to their natural complexity do not lend themselves well to parallelization, making them ill suited for parallel and distributed data analysis frameworks (Dean and Ghemawat 2004; Isard et al. 2007).

Given the efficiency benefits of incremental computation, an interesting question is whether it would be possible to achieve these benefits without requiring the design and implementation of incremental algorithms on an ad hoc basis. Previous work on systems like Incoop (Bhatotia et al. 2011a,b), Nectar (Gunda et al. 2010), HaLoop (Bu et al. 2010), DryadInc (Isard et al. 2007), or Ciel (Murray et al. 2011) shows that such gains are possible to obtain in a transparent way, i.e., without changing the original (single-pass) data analysis code. However, these systems

resort to the memoization of sub-computations from previous runs, which still requires time proportional to the size of the whole data rather than the change itself. Furthermore, these systems are meant to support arbitrary changes to the input and as such do not take advantage of the predictability of changes in sliding window computations to improve the timeliness of the results.

In this paper, we propose SLIDER, a system for incremental sliding window computations where the work performed by incremental updates is proportional to the size of the changes in the window (the “*delta*”) rather than the whole data. In SLIDER, the programmer expresses the computation corresponding to the analysis using either MapReduce (Dean and Ghemawat 2004) or another dataflow language that can be translated to the MapReduce paradigm (e.g., Hive (Foundation 2017) or Pig (Olston et al. 2008)). This computation is expressed by assuming a static, unchanging input window. The system then guarantees the automatic and efficient update of the output as the window slides. The system makes no restrictions on how the window slides, allowing it to shrink on one end and to grow on the other end arbitrarily (though as we show more restricted changes lead to simpler algorithms and more efficient updates). SLIDER thus offers the benefits of incremental computation in a fully transparent way.

Our approach to automatic incrementalization is based on the principles of self-adjusting computation (Acar 2005; Acar et al. 2009), a technique from the programming-languages community that we apply to sliding window computations. In self-adjusting computation, a *dynamic dependency graph* records the control and data dependencies of the computation, so that a *change-propagation algorithm* can update the graph as well as the data whenever the data changes. One key contribution of this paper is a set of novel data structures to represent the dependency graph of sliding window computations that allow for performing work proportional primarily to the size of the slide, incurring only a logarithmic – rather than linear – dependency to the size of the window.

To the best of our knowledge, this is the first technique for updating distributed incremental computations that achieves efficiency of this kind.

We further improve the proposed approach by designing a *split processing model* that takes advantage of structural properties of sliding window computations to improve response times. Under this model, we split the application processing into two parts: a foreground and a background processing. The foreground processing takes place right after the update to the computation window and minimizes the processing time by combining new data with a precomputed intermediate result. The background processing takes place after the result is produced and returned, paving the way for an efficient foreground processing by precomputing the intermediate result that will be used in the next incremental update.

We implemented SLIDER by extending Hadoop and evaluate its effectiveness by applying it to a variety of micro-benchmarks and applications and consider three real-world cases (Bhatotia et al. 2012b, 2014). Our experiments show that SLIDER can deliver significant performance gains while incurring only modest overheads for the initial run (non-incremental preprocessing run).

Background and Overview

In this section, we present some background and an overview of the design of SLIDER.

Self-Adjusting Computation

Self-adjusting computation is a field that studies ways to incrementalize programs automatically, without requiring significant changes to the code base (Acar 2005; Bhatotia 2015). For automatic incrementalization, in self-adjusting computations, the system constructs and maintains a *dynamic dependency graph* that contains the input data to a program, all sub-computations, and the data and control dependencies in between, e.g., which outputs of sub-computations are used as inputs to other sub-computations, which sub-computations are created by another.

The dynamic dependency graph enables a *change propagation* algorithm to update the computation and the output by propagating changes through the dependency graph, re-executing all sub-computations transitively affected by the change, reusing unaffected computations, and deleting obsolete sub-computations which no longer take place. The change propagation algorithm has been shown to be effective for a broad class of computations called *stable* and has even helped solve algorithmically sophisticated problems in a more efficient way than more ad hoc approaches (e.g., Acar 2010 and Sümer et al. 2011.)

Design Overview

Our primary goal is to design techniques for automatically incrementalizing sliding window computation to efficiently update their outputs when the window slides. We wish to achieve this without requiring the programmer to change any of the code, which is written assuming that the windows do not slide, i.e., the data remains static. To this end, we apply the principles of self-adjusting computation to develop a way of processing data that leads to stable computation, i.e., one whose overall dependency structure does not change dramatically when the window slides. In this paper, we apply this methodology to the MapReduce model (Dean and Ghemawat 2004), which enables expressive, fault-tolerant computations on large-scale data, thereby showing that the approach can be practical in modern data analysis systems. Nonetheless, the techniques we propose can be applied more generally, since we only require that the computation can be divided in a parallel, recursive fashion. In fact, the techniques that we propose can be applied to other large-scale data analysis models such as Dryad (Isard et al. 2007), Spark (Zaharia et al. 2012), and Pregel (Malewicz et al. 2010), which allow for recursively breaking up a computation into sub-computations.

Assuming that the reader is familiar with the MapReduce model, we first outline a basic design and then identify the limitations of this basic approach and describe how we overcome them. For simplicity, we assume that each job includes a single Reduce task (i.e., all mappers output

tuples with the same key). By symmetry, this assumption causes no loss of generality; in fact our techniques and implementation apply to multiple keys and reducers.

The basic design. Our basic design corresponds roughly to the scheme proposed in prior work, Incoop (Bhatotia et al. 2011a,b), where new data items are appended at the end of the previous window and old data items are dropped from the beginning. To update the output incrementally, we launch a Map task for each new “split” (a partition of the input that is handled by a single Map task) that holds new data and reuse the results of Map tasks operating on old but live data. We then feed the newly computed results together with the reused results to the Reduce task to compute the final output.

Contraction trees and change propagation. The basic design suffers from an important limitation: it cannot reuse any work of the Reduce task. This is because the Reduce task takes to input all values for a given key ($< K_i >$, $< V_1, V_2, V_3, \dots, V_n >$), and therefore a single change triggers a recomputation of the entire Reduce task. We address this by organizing the Reduce tasks as a contraction tree and proposing a change propagation algorithm that can update the result by performing traversals on the contraction tree, while maintaining it balanced (low-depth) and thus guaranteeing efficiency.

Contraction trees (used in Incoop (Bhatotia et al. 2011b), Camdoop (Costa et al. 2012), iMR (Logothetis et al. 2011)) leverage *Combiner functions* of MapReduce to break a Reduce task into smaller sub-computations that can be performed in parallel. Combiner functions originally aim at saving bandwidth by offloading parts of the computation performed by the Reduce task to the Map task. To use Combiners, the programmer specifies a separate Combiner function, which is executed on the machine that runs the Map task, and performs part of the work done by the Reduce task in order to preprocess various $\langle \text{key}, \text{value} \rangle$ pairs, merging them into a smaller number of pairs. The Combiner function

takes both as an input and output types a sequence of $\langle \text{key}, \text{value} \rangle$ pairs. This new usage of Combiners is compatible with the original Combiner interface but requires associativity for grouping different Map outputs. To construct a contraction tree, we break up the work done by the (potentially large) Reduce task into many applications of the Combiner functions. More specifically, we split the Reduce input into small groups and apply the Combine to pairs of groups recursively in the form of a balanced tree until we have a single group left. We apply the Reduce function to the output of the last Combiner.

When the window slides, we employ a *change-propagation algorithm* to update the result. Change propagation runs the Map on the newly introduced data and propagates the results of the Map tasks through the contraction tree that replaced the Reduce task. One key design challenge is how to organize the contraction tree so that this propagation requires minimal amount of time. We overcome this challenge by designing contraction trees and the change propagation algorithm such that change propagation always guarantees that the contraction trees remain balanced, guaranteeing low depth. As a result, change propagation can perform efficient updates by traversing only through a short path of the contraction tree. In this approach, performing an update not only updates the data but also the structure of the contraction tree (so that it remains balanced), guaranteeing that updates remain efficient regardless of the past history of window slides.

In addition to contraction trees and change propagation, we also present a *split processing* technique that takes advantage of the structural properties of sliding window computation to improve response times. Split processing splits the total work into background tasks, which can be performed when no queries are executing, and foreground tasks which must be performed in order to respond to a query correctly. By preprocessing certain computation in the background, split processing can improve performance significantly (up to 40% in our experiments).

The techniques that we present here are fully general: they apply to sliding window computa-

tions where the window may be resized arbitrarily by adding as much new data at the end and removing as much new data from the beginning as desired. As we describe, however, more restricted updates where, for example, the window is only expanded by append operations (*append-only*) or where the size of the window remains the same (*fixed-size windows*) lead to a more efficient and simpler design, algorithms, and implementation.

Slider Architecture

In this section, we give an overview of our implementation. Its key components are described in the following.

Implementation of self-adjusting trees. We implemented our prototype of SLIDER based on Hadoop. We implemented the three variants of the self-adjusting contraction tree by inserting an additional stage between the shuffle stage (where the outputs of Map tasks are routed to the appropriate Reduce task) and the sort stage (where the inputs to the Reduce task are sorted). To prevent unnecessary data movement in the cluster, the new contraction phase runs on the same machine as the Reduce task that will subsequently process the data.

SLIDER maintains a distributed dependency graph from one run to the next and pushes the changes through the graph by recomputing subcomputations affected by the changed input. For this purpose, we rely on a memoization layer to remember the inputs and outputs of various tasks and the nodes of the self-adjusting contraction trees. A shim I/O layer provides access to the memoization layer. This layer stores data in an in-memory distributed data cache, which provides both low-latency access and fault tolerance.

Split processing knob. SLIDER implements the split processing capability as an optional step that is run offline on a best effort basis. This stage is scheduled during low cluster utilization periods as a background task. It runs as a distributed data processing job (similar to MapReduce) with only one offline preprocessing stage.

In-memory distributed data caching. To provide fast access to memoized results, we designed an in-memory distributed data caching layer. Our use of in-memory caching is motivated by two observations: First, the number of subcomputations that need to be memoized is limited by the size of the sliding window. Second, main memory is generally underutilized in data-centric computing (Ananthanarayanan 2012). The distributed in-memory cache is coordinated by a master node, which maintains an index to locate the memoized results. The master implements a simple cache replacement policy, which can be changed according to the workload characteristics. The default policy is the Least Recently Used (LRU).

Fault-tolerance. Storing memoized results in an in-memory cache is beneficial for performance but can lead to reduced cache effectiveness when machines fail, as it requires unnecessary recomputation (especially for long running jobs). We therefore conduct a background replication of memoized results to provide fault tolerance by creating two replicas of each memoized result (similar to RAMCloud (Ongaro et al. 2011)). This way fault tolerance is handled transparently: when a new task wants to fetch a memoized result, it reads that result from one of the replicas. To ensure that the storage requirements remain bounded, we developed a garbage collection algorithm that frees the storage used by results that fall out of the current window.

Scheduler modifications. The implementation of SLIDER modifies Hadoop’s scheduler to become aware of the location of memoized results. Hadoop’s scheduler chooses any available node to run a pending Reduce task, only taking locality into account when scheduling Map tasks by biasing toward the node holding the input. SLIDER’s scheduler adapts previous work in data locality scheduling (Bhatotia et al. 2011b; Wieder et al. 2010a,b, 2012), by attempting to run Reduce tasks on the machine that contains the memoized outputs of the combiner phase. When that target machine is overloaded, it migrates tasks from the overloaded node to another node, including

the relevant memoized results. Task migration is important to prevent a significant performance degradation due to straggler tasks (Zaharia et al. 2008).

Dataflow query processing. As SLIDER is based on Hadoop, computations can be implemented using the MapReduce programming model. While MapReduce is gaining in popularity, many programmers are more familiar with query interfaces, as provided by SQL or LINQ. To ease the adoption of SLIDER, we also provide a dataflow query interface that is based on Pig (Olston et al. 2008). Pig consists of a high-level language similar to SQL and a compiler, which translates Pig programs to sequences of MapReduce jobs. As such, the jobs resulting from this compilation can run in SLIDER without adjustment. This way, we transparently run Pig-based sliding window computations in an incremental way.

Related Work

Next, we present a survey of work in the areas of sliding window and incremental processing.

Dynamic and data-streaming algorithms. This class of algorithms is designed to efficiently handle changing input data. Several surveys discuss the vast literature on dynamic algorithms, e.g., (Chiang and Tamassia 1992; Ramalingam and Reps 1993; Demetrescu et al. 2004; Babcock 2002). Despite their efficiency, dynamic algorithms are difficult to develop and specific to a use case: their adaptation to other applications is either not simple or not feasible.

Programming language-based approaches. In programming languages, researchers developed incremental computation techniques to achieve automatic incrementalization (Ramalingam and Reps 1993; Acar 2005). The goal is to incrementalize programs automatically without sacrificing efficiency. Recent advances on self-adjusting computation made significant progress toward this goal by proposing general-purpose

techniques that can achieve optimal update times (Acar 2005). This work, however, primarily targets sequential computations. The recent work on iThreads (Bhatotia et al. 2015) supports parallel incremental computation. In contrast, we developed techniques that are specific to sliding window computations and operate on big data by adapting the principles of self-adjusting computation for a large-scale parallel and distributed execution environment.

Database systems. There is substantial work from the database community on incrementally updating a database view (i.e., a predetermined query on the database) as the database contents change (Ceri and Widom 1991). database view (Ceri and Widom 1991). In contrast, our focus on the MapReduce model, with a different level of expressiveness, and on sliding window computations over big data brings a series of different technical challenges.

Distributed systems. There is some work on incremental large-scale processing of unstructured data sets (Peng and Dabek 2010; Gunda et al. 2010; Logothetis et al. 2010; Bhatotia et al. 2011a,b, 2012a; Krishnan et al. 2016). SLIDER is designed to operate at the same scale and shares some characteristics with this work, such as including a simple programming model, transparent data parallelization, fault tolerance, and scheduling. Furthermore, it builds on some concepts from this prior work, namely, the idea of Incoop (Bhatotia et al. 2011b) to break up the work of Reduce task using Combiner functions organized in a tree. There are two important distinctions to these previous proposals. First, prior proposals such as Incoop use memoization instead of change propagation: when the data to a MapReduce computation changes, Incoop scans the whole input data again and performs all the steps of the computation except that it can reuse results from Map and Combine tasks stored via memoization. In contrast, each run in SLIDER knows only about the new data that was introduced to the window and the old data that was dropped and only has to examine and recompute the subset of the computation that is

transitively affected by those changes. Second, our design includes several novel techniques that are specific to sliding window computations which are important to improve the performance for this mode of operation. The follow-up work on IncApprox (Krishnan et al. 2016) extends the approach of Slider to support incremental approximate computing for sliding window analytics. In particular, IncApprox combines incremental computation and approximates computation (Quoc et al. 2017a,b,c,d).

Batched stream processing. Stream processing engines (He et al. 2010; Condie et al. 2010; Olston et al. 2011) model the input data as a stream, with data-analysis queries being triggered upon bulk appends to the stream. These systems are designed for append-only data processing, which is only one of the cases we support. Compared to Comet (He et al. 2010) and Nova (Olston et al. 2011), SLIDER avoids the need to design a dynamic algorithm, thus preserving the transparency relative to single-pass non-incremental data analysis. Hadoop online (Condie et al. 2010) is transparent but does not attempt to break up the work of the Reduce task, which is one of the key contributions and sources of performance gains of our work.

Conclusion

In this chapter, we present techniques for incrementally updating sliding window computations as the window slides. The idea behind our approach is to structure distributed data-parallel computations in the form of balanced trees that can perform updates in asymptotically sublinear time, thus much more efficiently than recomputing from scratch. We present several algorithms for common instances of sliding window computations; describe the design of a system, SLIDER (Bhatotia et al. 2012b, 2014), that uses these algorithms; and present an implementation along with an extensive evaluation. Our evaluation shows that (1) SLIDER is effective on a broad range of applications, (2) SLIDER drastically improves performance compared to

the recomputing from scratch, and (3) SLIDER significantly outperforms several related systems. These results show that some of the benefits of incremental computation can be realized automatically without requiring programmer-controlled hand incrementalization. The asymptotic efficiency analysis of self-adjusting contraction trees is available (Bhatotia 2016).

References

- Acar UA (2005) Self-adjusting computation. PhD thesis, Carnegie Mellon University
- Acar UA, Blelloch GE, Blume M, Harper R, Tangwongsan K (2009) An experimental analysis of self-adjusting computation. ACM Trans Program Lang Syst (TOPLAS) 32(1):1–53
- Acar UA, Cotter A, Hudson B, Türkoğlu D (2010) Dynamic well-spaced point sets. In: Proceedings of the 26th annual symposium on computational geometry (SoCG)
- Ananthanarayanan G, Ghodsi A, Wang A, Borthakur D, Shenker S, Stoica I (2012) PACMan: coordinated memory caching for parallel jobs. In: Proceedings of the 9th USENIX conference on networked systems design and implementation (NSDI)
- Apache Software Foundation. Apache Hive (2017)
- Babcock B, Datar M, Motwani R, O’Callaghan L (2002) Sliding window computations over data streams. Technical report
- Bhatotia P (2015) Incremental parallel and distributed systems. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS)
- Bhatotia P (2016) Asymptotic analysis of self-adjusting contraction trees. CoRR, abs/1604.00794
- Bhatotia P, Wieder A, Akkus IE, Rodrigues R, Acar UA (2011a) Large-scale incremental data processing with change propagation. In: Proceedings of the conference on hot topics in cloud computing (HotCloud)
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquini R (2011b) Incoop: MapReduce for incremental computations. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Bhatotia P, Rodrigues R, Verma A (2012a) Shredder: GPU-accelerated incremental storage and computation. In: Proceedings of USENIX conference on file and storage technologies (FAST)
- Bhatotia P, Dischinger M, Rodrigues R, Acar UA (2012b) Slider: incremental sliding-window computations for large-scale data analysis. Technical report MPI-SWS-2012-004, MPI-SWS. <http://www.mpi-sws.org/tr/2012-004.pdf>
- Bhatotia P, Acar UA, Junqueira FP, Rodrigues R (2014) Slider: incremental sliding window analytics. In: Proceedings of the 15th international middleware conference (Middleware)

- Bhatotia P, Fonseca P, Acar UA, Brandenburg B, Rodrigues R (2015) iThreads: a threading library for parallel incremental computation. In: Proceedings of the 20th international conference on architectural support for programming languages and operating systems (ASPLOS)
- Bu Y, Howe B, Balazinska M, Ernst MD (2010) HaLoop: efficient iterative data processing on large clusters. In: Proceedings of the international conference on very large data bases (VLDB)
- Ceri S, Widom J (1991) Deriving production rules for incremental view maintenance. In: Proceedings of the international conference on very large data bases (VLDB)
- Chiang Y-J, Tamassia R (1992) Dynamic algorithms in computational geometry. In: Proceedings of the IEEE
- Condie T, Conway N, Alvaro P, Hellerstein JM, Elmeleegy K, Sears R (2010) MapReduce online. In: Proceedings of the 7th USENIX conference on networked systems design and implementation (NSDI)
- Costa et al (2012) Camdoop: exploiting in-network aggregation for big data applications. In: Proceedings of the 9th USENIX conference on networked systems design and implementation (NSDI)
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Demetrescu C, Finocchi I, Italiano G (2004) Handbook on data structures and applications. Chapman & Hall/CRC, Boca Raton
- Gunda PK, Ravindranath L, Thekkath CA, Yu Y, Zhuang L (2010) Nectar: automatic management of data and computation in datacenters. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- He B, Yang M, Guo Z, Chen R, Su B, Lin W, Zhou L (2010) Comet: batched stream processing for data intensive distributed computing. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of the ACM European conference on computer systems (EuroSys)
- Krishnan DR, Quoc DL, Bhatotia P, Fetzer C, Rodrigues R (2016) IncApprox: a data analytics system for incremental approximate computing. In: Proceedings of the 25th international conference on world wide web (WWW)
- Logothetis D, Olston C, Reed B, Web K, Yocum K (2010) Stateful bulk processing for incremental analytics. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Logothetis D, Trezzo C, Webb KC, Yocum K (2011) In-situ MapReduce for log processing. In: Proceedings of the 2011 USENIX conference on USENIX annual technical conference (USENIX ATC)
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Murray DG, Schwarzkopf M, Smowton C, Smith S, Madhavapeddy A, Hand S (2011) CIEL: a universal execution engine for distributed data-flow computing. In: Proceedings of the 8th USENIX conference on networked systems design and implementation (NSDI)
- Olston C et al (2008) Pig Latin: a not-so-foreign language for data processing. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Olston C et al (2011) Nova: continuous pig/hadoop workflows. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- Ongaro D, Rumble SM, Stutsman R, Ousterhout J, Rosenblum M (2011) Fast crash recovery in RAMCloud. In: Proceedings of the twenty-third ACM symposium on operating systems principles (SOSP)
- Peng D, Dabek F (2010) Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017a) Privacy preserving stream analytics: the marriage of randomized response and approximate computing. <https://arxiv.org/abs/1701.05403>
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017b) PrivApprox: privacy-preserving stream analytics. In: Proceedings of the 2017 USENIX conference on USENIX annual technical conference (USENIX ATC)
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017c) Approximate stream analytics in Apache flink and Apache spark streaming. CorRR, abs/1709.02946
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017d) StreamApprox: approximate computing for stream analytics. In: Proceedings of the international middleware conference (Middleware)
- Ramalingam G, Reps T (1993) A categorized bibliography on incremental computation. In: Proceedings of the ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)
- Sümer O, Acar UA, Ihler A, Mettu R (2011) Adaptive exact inference in graphical models. J Mach Learn
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010a) Brief announcement: modelling MapReduce for optimal execution in the cloud. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing (PODC)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010b) Conductor: orchestrating the clouds. In: Proceedings of the 4th international workshop on large scale distributed systems and middleware (LADIS)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2012) Orchestrating the deployment of computations in the cloud with conductor. In: Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI)
- Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I (2008) Improving MapReduce performance in hetero-

- geneous environments. In: Proceedings of the USENIX conference on operating systems design and implementation (OSDI)
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation (NSDI)

Indexing

Jianzhong Qi and Rui Zhang
The University of Melbourne, Melbourne, VIC,
Australia

Synonyms

[Big spatial data access methods](#); [Indexing big spatial data](#)

Definitions

Consider a set of n data objects $O = \{o_1, o_2, \dots, o_n\}$. Each object is associated with a d -dimensional vector representing its coordinate in a d -dimensional space ($d \in \mathbb{N}_+$). *Indexing* such a set of data is to organize the data in a way that provides fast access to the data, for processing spatial queries such as *point queries*, *range queries*, *kNN queries*, *spatial join queries*, etc.

Overview

The rapid growth of *location-based services* (LBS) has accumulated a massive amount of spatial data such as user GPS coordinates, which calls for efficient indexing structures to provide fast access to such data. Typical applications of spatial data include digital mapping services and location-based social networks, where spatial queries are issued by users such as *finding restaurants within 3 kilometers around*

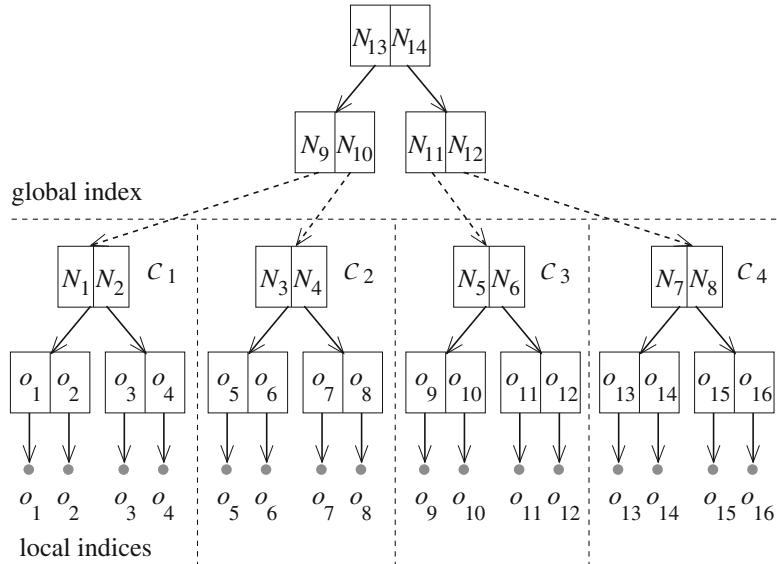
Alice or finding other users within 300 meters around Alice. Spatial indices have been studied extensively to support such queries. In the big data era, there may be millions of spatial queries and places of interests to be processed at the same time. This poses new challenges in both scalability and efficiency of spatial indexing techniques. Parallel spatial index structures are developed to address these challenges, which are summarized in the following section.

Key Research Findings

Traditional spatial indexing techniques can be classified into *space partitioning* techniques and *data partitioning* techniques. Space partitioning techniques such as *quad trees* (Finkel and Bentley 1974) partition the data space into nonoverlapping partitions where data objects in the same partition are indexed together. Data partitioning techniques such as *R-trees* (Guttman 1984) partition the data set directly where data objects nearby are grouped into the same partition and indexed together. We focus on recent developments of the indexing techniques. Readers interested in the traditional spatial indices are referred to a detailed survey on this topic by Gaede and Günther (1998).

Spatial indices have been extended to parallel computing environments to cope with the scalability and efficiency issues. For example, the R-trees have been implemented over a shared-nothing (client-server) architecture. Koudas et al. (1996) store the inner nodes of an R-tree on a server and the leaf nodes on clients. In this architecture, the inner nodes stored on the server forms a *global index*. At query processing, the server uses this index to prune the search space and locate the clients that contain the data objects being queried. Schnitzer and Leutenegger (1999) further create local R-trees on clients. This forms a two-level index structure where the global index is hosted on the server while the *local indices* are hosted on the clients. This is illustrated in Fig. 1, where there is a global index stored on the server and four local indices stored on four client machines C_1, C_2, C_3 , and C_4 .

Indexing, Fig. 1
Two-level distributed indexing



More recent work on parallelizing spatial indices uses the *MapReduce* and the *Spark* frameworks which have become standard parallel computation frameworks in the past decade. These frameworks hide the parallelism details such as synchronization and simplify parallel computation into a few generic operations (e.g., *Map* and *Reduce*). The two-level index structure as described above is still commonly used, although some of the techniques proposed only use either the global index or the local indices. We describe a few representative index structures using these two frameworks next.

MapReduce-Based Indexing

Hadoop-GIS (Aji et al. 2013) is a *Hadoop* (an open-source implementation of the MapReduce framework)-based spatial data warehousing system. It builds a two-level (global and local) index structure over the data objects to support parallel processing of spatial queries. The global index is pre-built by first partitioning the data space with a regular grid and then further partitioning every grid cell containing more than a predefined number of data objects n_{\max} into two equi-sized cells at the middle recursively until each cell contains no more than n_{\max} objects. Data objects in the same cell are associated with the same unique id and stored in the same block. Multiple

cells are grouped into a block for storage to suit the block size in the *HDFS* (a distributed file system used by Hadoop). The *minimum bounding rectangles* (MBR) of the grid cells are stored in the global index. A local index is built at query time over the data objects assigned to the same machine for query processing.

SpatialHadoop (Eldawy and Mokbel 2013) extends Hadoop to provide built-in support for spatial queries. It also uses a global index and a local index. Grid-based and R-tree-based indices can both be used for the global index, where the grid cells and leaf nodes, respectively, correspond to partitions that are stored in HDFS blocks. A local index is built for each partition, which can also use grid or R-tree-based indices. *AQWA* (Aly et al. 2015) builds a global index only, which uses the k-d tree. This index also stores the number of data points of each partition to help query processing.

A few other studies (e.g., Nishimura et al. 2011; Hsu et al. 2012; Zhang et al. 2014) use *space-filling curves* such as the *Z-curve* (Orenstein and Merrett 1984) to map multidimensional coordinates to one-dimensional values. This enables storing the data objects into *NoSQL* databases such as *HBase*, where the mapped coordinate value and the id of a data object are stored as a *key-value* pair. Data update

and spatial queries can then be handled via the APIs of NoSQL databases.

When a hierarchical index such as the R-tree or the k-d tree is built with MapReduce, dynamic index building procedures that insert the data objects into the index individually lack efficiency. A common assumption is that the entire data set is available. In this case, the entire index structure over the data set can be *bulk-loaded* at once. A straightforward approach is to bulk-load the hierarchical index level by level, where each level incurs a round of MapReduce computation (Achakeev et al. 2012). Sampling has been used to reduce the number of MapReduce rounds needed. Papadopoulos and Manolopoulos (2003) propose a generic procedure for parallel spatial index bulk-loading. This procedure uses sampling to estimate the data distribution, which guides the partition of the data space. Data objects in the same partition are assigned to the same machine for index building. SpatialHadoop uses a similar idea for R-tree bulk-loading, where an R-tree is built on a sample data set and the MBR of the leaf nodes are used for data partitioning. Agarwal et al. (2016) also use a sampling-based technique but bulk-load k-d trees. This technique involves multiple rounds of sampling. Each round uses a sample data set small enough to be processed by a single machine. A k-d tree is built on the sample data set, and data objects are assigned to the partitions created by the k-d tree. For the partitions that contain too many data objects to be stored together, another round of sampling- and k-d tree-based partitioning is performed. Repeating the procedure above, it takes $O(\text{poly} \log_s n)$ MapReduce rounds to bulk-load a k-d tree, where s denotes the number of data points that can be processed by a single machine in the MapReduce cluster.

Spark-Based Indexing

Spark models a data set as a *resilient distributed dataset* (RDD) and stores it across machines in the cluster. Spatial indices on Spark focus on optimizing with the RDD storage architecture.

GeoSpark (Yu et al. 2015) builds a local index such as a quad-tree on-the-fly for the data objects in each RDD partition, while the partitions are

created by a uniform grid partitioning over the data space. *SpatialSpark* (You et al. 2015) also builds a local index for data objects in each RDD partition, which can be stored together with the data objects in the RDD partition. SpatialSpark supports binary space partitioning and tile partitioning in addition to uniform grid partitioning over the data space. *STARK* (Hagedorn et al. 2017) uses R-trees for the local indices while uniform grid partitioning and cost-based binary space partitioning for creating the partitions. *Simba* (Xie et al. 2016) uses global and local indices (e.g., R-trees). The global index is held in-memory in the master node, while the local indices are held in each RDD partition.

Examples of Application

Spatial indexing techniques are used in spatial databases to support applications in both business and daily scenarios such as targeted advertising, digital mapping, and location-based social networking. In these applications, the locations (coordinates) of large sets of places of interests as well as users are stored in a spatial database. Spatial indices are built on top of such data to provide fast data access, facilitating spatial queries such as *finding customers within 100 meters of a shopping mall* (to push shopping vouchers), *finding the nearest restaurants for Alice*, and *finding users living in nearby suburbs who share some common interests* (for friendship recommendations).

Future Directions for Research

The indexing techniques discussed above do not consider handling location data with frequent updates. Such data is becoming more and more common with the increasing popularity of positioning system-enabled devices such as smart phones, which enables tracking and querying the continuously changing locations of users or data objects of interest (Zhang et al. 2012; Ward et al. 2014; Li et al. 2015). How to update the spatial indices and to record multiple versions of the

data (Lomet et al. 2008) with a high frequency is nontrivial. While the parallel frameworks such as MapReduce scale well to massive data, the indices stored in the distributed data storage may not be updated as easily as indices stored in a standalone machine. A periodic rebuilt of the indices is an option, but may not keep the indices updated between rebuilds, and hence may provide inaccurate query answers. More efficiency index update techniques are in need. Another important direction to pursue is efficient indexing techniques to support spatial data mining. Many data mining algorithms are computational intensive, which require iterative accesses to (high-dimensional) data. Spatial indices designed to cope with high-dimensional data (Zhang et al. 2004; Jagadish et al. 2005) and the special accessing patterns of such algorithms would have a significant impact in improving the usability of such algorithms.

Cross-References

- [Query Processing: Computational Geometry](#)
- [Query Processing: Joins](#)
- [Query Processing – kNN](#)

References

- Achakeev D, Seidemann M, Schmidt M, Seeger B (2012) Sort-based parallel loading of r-trees. In: Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data, pp 62–70
- Agarwal PK, Fox K, Munagala K, Nath A (2016) Parallel algorithms for constructing range and nearest-neighbor searching data structures. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems (PODS), pp 429–440
- Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J (2013) Hadoop gis: a high performance spatial data warehousing system over mapreduce. Proc VLDB Endow 6(11):1009–1020
- Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) Aqwa: adaptive query workload aware partitioning of big spatial data. Proc VLDB Endow 8(13):2062–2073
- Eldawy A, Mokbel MF (2013) A demonstration of spatial-hadoop: an efficient mapreduce framework for spatial data. Proc VLDB Endow 6(12):1230–1233
- Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Informatica 4(1):1–9
- Gaede V, Günther O (1998) Multidimensional access methods. ACM Comput Surv 30(2):170–231
- Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data (SIGMOD), pp 47–57
- Hagedorn S, Götze P, Sattler K (2017) Big spatial data processing frameworks: feature and performance evaluation. In: Proceedings of the 20th international conference on extending database technology (EDBT), pp 490–493
- Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC (2012) Key formulation schemes for spatial index in cloud data managements. In: 2012 IEEE 13th international conference on mobile data management (MDM), pp 21–26
- Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) Idistance: an adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
- Koudas N, Faloutsos C, Kamel I (1996) Declustering spatial databases on a multi-computer architecture. In: Proceedings of the 5th international conference on extending database technology: advances in database technology (EDBT), pp 592–614
- Li C, Gu Y, Qi J, Zhang R, Yu G (2015) A safe region based approach to moving KNN queries in obstructed space. Knowl Inf Syst 45(2):417–451
- Lomet D, Hong M, Nehme R, Zhang R (2008) Transaction time indexing with version compression. Proc VLDB Endow 1(1):870–881
- Nishimura S, Das S, Agrawal D, Abbadi AE (2011) Md-hbase: a scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 2011 IEEE 12th international conference on mobile data management (MDM), vol 01, pp 7–16
- Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on principles of database systems (PODS), pp 181–190
- Papadopoulos A, Manolopoulos Y (2003) Parallel bulk-loading of spatial data. Parallel Comput 29(10): 1419–1444
- Schnitzer B, Leutenegger ST (1999) Master-client r-trees: a new parallel r-tree architecture. In: Proceedings of the 11th international conference on scientific and statistical database management (SSDBM), pp 68–77
- Ward PGD, He Z, Zhang R, Qi J (2014) Real-time continuous intersection joins over large sets of moving objects using graphic processing units. VLDB J 23(6):965–985
- Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: efficient in-memory spatial analytics. In: Proceedings of the 2016 SIGMOD international conference on management of data (SIGMOD), pp 1071–1085

- You S, Zhang J, Gruenwald L (2015) Large-scale spatial join query processing in cloud. In: Proceedings of the 31st IEEE international conference on data engineering workshops, pp 34–41
- Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems (SIGSPATIAL), pp 70:1–70:4
- Zhang R, Ooi BC, Tan KL (2004) Making the pyramid technique robust to query types and workloads. In: Proceedings of the 20th international conference on data engineering (ICDE), pp 313–324
- Zhang R, Qi J, Lin D, Wang W, Wong RC (2012) A highly optimized algorithm for continuous intersection join queries over moving objects. VLDB J 21(4):561–586
- Zhang R, Qi J, Stradling M, Huang J (2014) Towards a painless index for spatial objects. ACM Trans Database Syst 39(3):19:1–19:42

Indexing Big Spatial Data

► Indexing

Indexing for Graph Query Evaluation

George Fletcher¹ and Martin Theobald²

¹Technische Universiteit Eindhoven, Eindhoven, Netherlands

²Université du Luxembourg, Luxembourg, Luxembourg

Definitions

Given a graph, an index is a data structure supporting a map from a collection of keys to a collection of elements in the graph. For example, we may have an index on node labels, which, given a node label as search key, facilitates accelerated access to all nodes of the graph having the given label. The evaluation of queries on graph databases is often facilitated by index data structures. An index can be the primary representation of the graph or can be a secondary access path to elements of the graph.

Overview

In this article, we give a succinct overview of the main approaches to indexing graphs for efficient graph query evaluation. We focus our discussion on exact query processing and do not consider lossy graph representations. Rather than aiming for an exhaustive survey, we illustrate each approach with select exemplars which highlight the main ideas of the approach.

Key Research Findings

There is a wealth of work on graph indexing, which can be organized along three directions: value indexing, reachability indexing, and compression and structural indexing. After introducing graph data and graph queries in the next section, we survey each of these approaches. We conclude the article with a discussion of issues in distribution and partitioning.

Graph Data and Queries

Data Model. The general data model we consider for this article is that of a *directed, labeled multigraph* $G = (V, E, L, \phi_V, \phi_E)$, consisting of a set of *vertices* V , a set of *edges* E over V , a set of *labels* L , a *vertex-labeling function* $\phi_V : V \mapsto L$, and an *edge-labeling function* $\phi_E : E \mapsto L$. For most applications, we assume ϕ_V to be bijective, i.e., all vertices have a distinct label (then also called “vertex identifier”), while ϕ_E is not necessarily injective nor surjective, i.e., edge labels may occur multiple times among edges.

Query Model. In analogy to our data model, we also consider a *subgraph pattern matching query* $G_Q = (V_Q, E_Q, L, Vars, \phi_Q)$ to be a labeled, directed multigraph, consisting of a set of *query vertices* V_Q , a set of *query edges* E_Q over V_Q , two disjoint sets of *labels* L and *variables* $Vars$ with $L \cap Vars = \emptyset$, and a labeling function $\phi_Q : V_Q \cup E_Q \mapsto L \cup Vars$. Also here, by

assuming unique vertex labels, we impose that $E_Q \subseteq V_Q \times (L \cup \text{Vars}) \times V_Q$.

Given a data graph $G = (V, E, L)$ and a query graph $G_Q = (V_Q, E_Q, L, \text{Vars})$, the semantics of evaluating G_Q on G consists of identifying all isomorphic subgraphs $G' = (V', E', L)$ of G , with $V' \subseteq V$, $E' \subseteq E \cap (V' \times L \times V')$, i.e., a bijection $f : V' \cup L \mapsto V_Q \cup L \cup \text{Vars}$ such that $(v_1, l, v_2) \in E' \implies (f(v_1), f(l), f(v_2)) \in E_Q$. For query evaluation, we impose the additional condition that for vertex and edge labels $l \in L$, it must hold that $f(l) = l$, i.e., the given labels in the query graph must be matched also by the data subgraph, such that only variables in the query graph are replaced by labels in the data subgraph. Some applications relax the semantics of query evaluation such that all homomorphic embeddings are constructed, i.e., f is not necessarily injective. Under homomorphic semantics, subgraph pattern matching queries can also be seen as *conjunctive queries* over relations consisting of edge-labeled adjacency lists with schema $V \times L \times V$ and be solved by a fixed number of joins over these relations.

RDF and SPARQL Directed labeled multi-graphs without repeating vertex labels are particularly well suited for indexing RDF data (RDF 2014) and for processing basic-graph-pattern (BGP) queries expressible in SPARQL 1.1 (SPA 2013). Due to the *unique-name assumption* and the corresponding usage of *unique resource identifiers* (URIs), both prevalent in the RDF data model, edge-labeled adjacency lists of the form $V \times L \times V$ resolve to subject-predicate-object (SPO) triples over the underlying RDF data graph. Most database-oriented indexing and query processing frameworks consider a collection of SPO triples either as a single large relation, or they horizontally partition the collection into multiple SO relations, one for each distinct predicate P (or “property” in RDF terminology). Moreover, various value- and grammar-based synopses and corresponding indexing techniques for RDF data have been developed recently. To further scale-out the indexing strategies to a distributed setting, both horizontal and vertical partitioning,

summarization, and related strategies have been developed, which we will cover in more detail in the following subsections.

SPARQL also supports the usage of complex-graph-pattern (CGP) queries with *labeled property paths* allows a user to formulate transitive reachability constraints among the query variables. Very few works so far focused on the combined processing of relational joins with additional graph-reachability predicates, e.g., Gubichev et al. (2013) and Sarwat et al. (2013). Przyjaciel-Zablocki et al. (2011) investigates an initial approach to evaluate property paths via MapReduce. Currently, only very few commercial RDF engines, such as Virtuoso (Erling and Mikhailov 2009), are available for processing property paths in SPARQL 1.1.

Value-Based Indexing

Value-based indexes are the simplest form of graph data structure, typically used as the primary representation of the graph in the database. Such indexes map values in the graph (node identifiers, edge identifiers, node labels, edge labels) to the nodes or edges associated with the search key. Among the centralized approaches, native RDF stores like 4store (Harris et al. 2009), SW-Store (Abadi et al. 2009), MonetDB-RDF (Sidiropoulos et al. 2008), and RDF-3X (Neumann and Weikum 2010a) have been carefully designed to keep pace with the growing scale of RDF collections. Efficient centralized architectures either employ various forms of horizontal (Neumann and Weikum 2010a; Harris et al. 2009) and vertical (Abadi et al. 2009; Sidiropoulos et al. 2008) partitioning schemes or apply sophisticated encoding schemes over the subjects’ properties (Atre et al. 2010; Yuan et al. 2013). We refer the reader to Luo et al. (2012) for a comprehensive overview of recent approaches for value-based indexing and query processing techniques for labeled graphs in the form of RDF.

Relational Approaches. The majority of existing RDF stores, both centralized and distributed, follow a relational approach toward

storing and indexing RDF data. By treating RDF data as a collection of triples, early systems like 3Store (Harris and Gibbins 2003) use relational databases to store RDF triples in a single large three-column table. This approach lacks scalability, as the query processing (then using SQL) involves many self-joins over this single table. Later approaches follow more sophisticated approaches for storing RDF triples. More recent approaches, such as SW-Store by Abadi et al. (2009), vertically partition RDF triples into multiple property tables. State-of-the-art centralized approaches, such as HexaStore (Weiss et al. 2008) and RDF-3X (Neumann and Weikum 2010a), employ index-based solutions by storing triples directly in B^+ -trees over multiple, redundant SPO permutations. Including all permutations and projections of the SPO attributes, this may result in up to 15 such B^+ -trees (Neumann and Weikum 2010a). Coupled with sophisticated statistics and query-optimization techniques, these centralized, index-based approaches still are very competitive as recently shown in Tsialiamanis et al. (2012).

Join-Order Optimization. Determining the optimal join-order for a query plan is arguably the main factor that impacts query processing performance of a relational engine. RDF-3X (Neumann and Weikum 2010a) thus performs an exhaustive plan enumeration in combination with a bottom-up DP algorithm and aggressive pruning in order to identify the best join-order. TriAD (Gurajada et al. 2014) adopts the DP algorithm, as described in Neumann and Weikum (2010a) and further adapts it to finding both the best exploration-order for the summary graph and the best join-order for the subsequent processing against the SPO indexes. Moreover, by including detailed distribution information and the ability to run multiple joins in parallel into the underlying cost model of the optimizer, TriAD is able to generate query plans that are more specifically tuned toward parallel execution than with a pure selectivity-based cost model.

Join-Ahead Pruning. Join-ahead pruning is a second main factor that influences the perfor-

mance of a relational query processor. In join-ahead pruning, triples that might not qualify for a join are pruned even before the actual join operator is invoked. This pruning of triples ahead of the join operators may thus save a substantial amount of computation time for the actual joins. Instead of the sideways information passing (SIP) strategy used in RDF-3X (Neumann and Weikum 2010a,b), which is a runtime form of join-ahead pruning, TriAD (Gurajada et al. 2014) employs a similar kind of pruning via graph compression, discussed below.

Reachability and Regular-Path Indexing

The reachability problem in directed graphs is one of the most fundamental graph problems and thus has been tackled by a plethora of centralized approaches; see Su et al. (2017) for a recent overview. All methods aim to find tradeoffs among query time and indexing space which, for a directed graph $G(V, E)$, are in between $\mathcal{O}(|V| + |E|)$ for both query time and space consumption when no indexes are used and $\mathcal{O}(1)$ query time and $\mathcal{O}(|V|^2)$ space consumption when the transitive closure is fully materialized. Distributed reachability queries for single-source, single-target queries have so far been considered in Fan et al. (2012b). For multisource, multi-target reachability queries, only comparably few centralized approaches have been developed (Gao and Anyanwu 2013; Then et al. 2014) that provide suitable indexing and processing strategies. Both are based on a notion of *equivalence sets* among graph vertices which effectively resolves to a preprocessing and indexing step of the data graph to predetermine these sets. Efficient centralized approaches, however, are naturally limited to the main memory of a single machine and usually do not consider a parallel, i.e., multi-threaded, execution of a reachability query. Then et al. (2014), on the other hand, focused on the *query-time optimization* of multisource BFS searches. Gubichev et al. (2013) have studied the use of reachability indexes for evaluation of richer property-path queries.

Compression for Query Evaluation

We next highlight three approaches for graph compression to accelerate graph query evaluation.

Structural Indexing

The first approach to graph compression considers graph summaries specifically tailored for a given query language. In particular, these methodologies propose *structural* indexing strategies, which identify characterizations of query language expressivity in terms of the (language independent) structure of the database instance.

Given a query language L , the general strategy proceeds in four steps. (1) A structural equivalence notion \equiv on graphs is identified which corresponds precisely with indistinguishability in L . Here we mean, given an arbitrary graph G and objects o_1 and o_2 of G , it holds that $o_1 \equiv o_2$ if and only if for every query $Q \in L$, it holds that either both or neither of o_1 and o_2 are in the evaluation of Q on G . In other words, objects are structurally equivalent if and only if they are indistinguishable by queries in L . Depending on L , objects might be nodes, edges, paths, or even more complex structures in G . Further, \equiv is independent of L in the sense that it can be computed based on the structure of G alone. (2) G is compressed under \equiv (i.e., the set of objects is partitioned by \equiv equivalence), giving the reduced instance G_{\equiv} . This reduced instance is the “ideal” basis for index data structures for accelerating evaluation of L on G . Indeed, elements of each equivalence class are identical from the perspective of L and so can be treated together in bulk during query evaluation. (3) Given a query $Q \in L$, it is rewritten and evaluated in terms of (an index data structure on) G_{\equiv} , and, finally (4) if necessary, the output of the previous step is post-processed to extract and materialize the complete results of Q evaluated on G .

Structural indexing only makes practical sense if the structural equivalence characterization is efficiently computable on large graphs. Core graph query languages such as the pattern

matching queries typically have impractical structural characterizations. The key challenge then is to identify relaxations of the problem which support efficient index construction and use for acceleration of query processing. There are two strategies which have been employed in this direction.

The first strategy is to identify languages which, on one hand, are rich enough to capture interesting queries and, on the other hand, have a tractable structural equivalence notion. This approach has been followed for the pattern matching queries and path queries, where expressive fragments of these languages have been identified with tractable structural characterizations. In particular, structural characterizations of fragments of the pattern matching queries have been identified by Fletcher et al. (2015) and Picalausa et al. (2012, 2014) and for fragments of path queries by Fan et al. (2012a) and Fletcher et al. (2015). Given a query Q in a full language, e.g., a pattern matching query, and a structural index I built for a particular fragment F of the language on instance G , evaluation of Q proceeds by (1) decomposing the query into subqueries in F , (2) evaluating these subqueries directly with I , and (3) further processing of these intermediate results to obtain the final result of evaluating Q on G . This methodology was first developed for navigational pattern matching queries on semi-structured data (Fletcher et al. 2009, 2016; Milo and Suciu 1999). Typically, the structural characterizations are variations of the well-known tractable structural notion of graph bisimulation. Effective methods for constructing bisimulation reductions of large acyclic and cyclic graphs have been developed for both external memory and distributed settings (Hellings et al. 2012; Luo et al. 2013b,a).

Instead of identifying “weaker” fragments of a query language in order to obtain practical structural characterizations, the second strategy which has been studied is to weaken the semantics of the query language itself. This approach has been developed by Fan et al. (2012a) for the case of the pattern matching queries. Here, adopting a

relaxed simulation-based semantics for pattern matching, query expressivity is captured by a tractable structural equivalence based on graph bisimulation. Although this relaxed semantics diverges from the standard semantics of pattern matching queries, which is typically based on the stronger notions of homomorphism or isomorphism, simulation-based semantics finds important applications, e.g., in analytics on social networks.

In general, structural indexing for graph query evaluation remains a relatively unexplored topic with strong potential for practical application.

Grammar-Based Compression

A second major approach to compression for query evaluation is based on using context-free grammars (CFG) for lossless graph summarization (Maneth and Peternek 2016).

Given a graph G , the basic idea is to build a CFG Γ_G which generates G . The gRePair method of Maneth and Peternek (2016) constructs Γ_G by repeatedly replacing “digrams” (i.e., pairs of connected edges in the graph) by edges labeled by nonterminals in Γ_G . Depending on the instance structure, the grammar Γ_G can be significantly smaller than G . Compression of this method, in terms of bits used per edge, has been shown experimentally to be competitive with state-of-the-art graph compressors.

In addition to high compression rates, Maneth and Peternek (2016) demonstrate good theoretical properties for query evaluation. In particular, adjacency (i.e., neighborhood) queries can be losslessly answered on Γ_G , reachability queries can be answered in linear time (in the size of Γ_G), and a regular path query Q can be answered in $O(|Q||\Gamma_G|)$ time. Hence, the speedup of path queries is proportional to the compression obtained with Γ_G .

In-depth empirical investigation of query evaluation performance on grammar-compressed graphs remains open. As with structural indexing, the study of grammar-based graph compression is a very promising and relatively unexplored approach for accelerated query evaluation.

Graph Dedensification

A third approach to compression-based acceleration of query evaluation is to introduce graph compression by modifying the structure of the graph itself. This strategy is orthogonal to, and can be used in combination with, the approaches discussed above.

Motivated by the large skew in degree-distributions in real-world complex networks, Maccioni and Abadi (2016) propose the use of graph “dedensification” to offset the negative impact of high-degree nodes during graph pattern matching query processing. Given a graph G , the basic idea is to identify and transform complete bipartite subgraphs in G , i.e., subgraphs whose node sets can be partitioned into subsets S and T such that there is an edge from each node in S to each node in T . In particular, for such subgraphs, (1) a new “compressor” node c is introduced; (2) all edges of the subgraph are removed; (3) for each node $s \in S$, an edge is added from s to c ; and (4) for each node $t \in T$, an edge is added from c to t . Hence, the introduction of the compressor nodes dedensifies the subgraph, by replacing a quadratic number of edges with a linear number of edges.

Given a dedensified graph, Maccioni and Abadi (2016) propose exact query evaluation strategies which are experimentally demonstrated to boost query evaluation performance by up to an order of magnitude.

The benefits of dedensification were demonstrated both in the absence of indexes and in combination with exhaustive valued-based indexing. The study of dedensification in combination with the compression strategies discussed above remains an interesting open problem.

Concluding our discussion of compression for query evaluation, we reiterate that, to our knowledge, the compression-based query evaluation solutions presented here have only been demonstrated in research prototypes. A major research challenge is to further study the systems challenges toward realizing these promising methods in graph database systems in practice.

Distribution and Partitioning

MapReduce. Based on the MapReduce paradigm, distributed engines like H-RDF-3X (Huang et al. 2011) and SHARD (Rohloff and Schantz 2011) horizontally partition an RDF collection over a number of compute nodes and employ Hadoop as communication layer for queries that span multiple nodes. H-RDF-3X (Huang et al. 2011) partitions an RDF graph into as many partitions as there are compute nodes. Then, a one- or two-hop replication is applied to index each of the local graphs via RDF-3X (Neumann and Weikum 2010b). Query processing in both systems is performed using iterative Reduce-side joins, where the Map phase performs selections and the Reduce phase performs the actual joins (Lin and Dyer 2010). Although such a setting works well for queries that scan large portions of the data graph, for less data-intensive queries, the overhead of iteratively running MapReduce jobs and scanning all – or at least large amounts – of the tuples during the Map phase may be significant. Even recent approaches like EAGRE (Zhang et al. 2013) that focus on minimizing I/O costs by carefully scheduling Map tasks and utilizing extensive data replication cannot completely avoid Hadoop-based joins in the case of longer-diameter queries or unexpected workloads.

Native Graph Approaches. Recently, a number of distributed approaches were proposed to store RDF triples in native graph format. The main motivation behind these approaches is that relational approaches might generate large intermediate relations which may hamper performance. Native graph engines typically employ adjacency lists as a basic building block for storing and processing RDF data. Moreover, by using sophisticated indexes, like gStore (Zou et al. 2011), BitMat (Atre et al. 2010), and TripleBit (Yuan et al. 2013), or by using graph exploration, like in Trinity.RDF (Zeng et al. 2013), these approaches prune many triples before invoking relational joins to finally generate the row-oriented results of a SPARQL query. Trinity.RDF thus represents

a new generation of native graph engines that are based on earlier ideas from systems such as Pregel (Malewicz et al. 2010) or Neo4j (Webber 2012). Other kinds of graph queries, such as reachability, shortest-paths, or random walks, are partly already included in the SPARQL 1.1 standard. Such queries are targeted by various graph engines, such as FERRARI (Seufert et al. 2013) or Spark’s GraphX (Xin et al. 2013).

Graph Partitioning. Apart from centralized engines like gStore (Zou et al. 2011), BitMat (Atre et al. 2010), and TripleBit (Yuan et al. 2013), Huang et al. (2011) also follow a graph partitioning approach over a distributed setting, where triples are assigned to different machines using a graph partitioner. Graph partitioning allows triples that are close to each other in the graph to be stored at the same machine, thus overcoming the randomness issued by the purely hashing-based partitioning schemes used in systems like SHARD (Rohloff and Schantz 2011).

Graph Exploration vs. Join Processing. To avoid the overhead of Hadoop-based joins, Trinity.RDF (Zeng et al. 2013) is based on a custom protocol based on the message passing interface (MPI) (Message Passing Interface Forum 2015) for internode communication. In Trinity.RDF, intermediate variable bindings are computed among all slave nodes via graph exploration, while the final results need to be enumerated at the single master node using a single-threaded, left-deep join over the intermediate variable bindings.

Distributed Reachability Queries. Very few works so far have been developed for distributed reachability queries. Fan et al. (2012b) recently discussed distributed, but single-source, single-target reachability algorithms for partitioned graphs and also provide performance guarantees. For a directed graph and given cut, Fan et al. (2012b) uses a partially iterative and partially indexing-based evaluation to find the reachability for a given pair of vertices over a classical master-slave architecture. Distributed graph engines, such as Google’s Pregel (Malewicz et al. 2010),

Berkeley's GraphX (Xin et al. 2013) (based on Spark Zaharia et al. 2010), Apache Giraph (Ching et al. 2015) and IBM's very recent Giraph++ (Tian et al. 2013), on the other hand, allow for the scalable processing of graph algorithms over massive, distributed data graphs. All of these provide generic API's for implementing various kinds of algorithms, including multisource, multi-target reachability queries. Graph indexing and query optimization, however, break the *node-centric computing* paradigm of Pregel and Giraph, where major amounts of the graph are successively shuffled through the network in each of the underlying MapReduce iterations (the so-called supersteps). Giraph++ is a very recent approach to overcome this overly myopic form of node-centric computing, which led to a new type of distributed graph processing that is coined *graph-centric computing* in Tian et al. (2013). By exposing intra-node state information as well as the internode partition structure to the local compute nodes, Giraph++ is a great step toward making these graph computations more context-aware. Further distributed approaches include Gurajada and Theobald (2016), Harbi et al. (2015), Peng et al. (2016), and Potter et al. (2016).

Cross-References

- ▶ [Graph Data Models](#)
- ▶ [Graph Partitioning: Formulations and Applications to Big Data](#)
- ▶ [Graph Query Languages](#)
- ▶ [Graph Representations and Storage](#)

References

- Abadi DJ, Marcus A, Madden SR, Hollenbach K (2009) SW-Store: a vertically partitioned DBMS for semantic web data management. VLDB J 18(2):385–406
- Atre M, Chaoji V, Zaki MJ, Hendler JA (2010) Matrix “Bit” loaded: a scalable lightweight join query processor for RDF data. In: WWW. ACM, pp 41–50
- Ching A, Edunov S, Kabiljo M, Logothetis D, Muthukrishnan S (2015) One trillion edges: graph processing at facebook-scale. PVLDB 8(12):1804–1815
- Erling O, Mikhailov I (2009) Virtuoso: RDF support in a native RDBMS. In: SWIM, pp 501–519
- Fan W, Li J, Wang X, Wu Y (2012a) Query preserving graph compression. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2012, Scottsdale, 20–24 May 2012, pp 157–168
- Fan W, Wang X, Wu Y (2012b) Performance guarantees for distributed reachability queries. PVLDB 5(11):1304–1315
- Fletcher G, Van Gucht D, Wu Y, Gyssens M, Brenes S, Paredaens J (2009) A methodology for coupling fragments of XPath with structural indexes for XML documents. Inf Syst 34(7):657–670
- Fletcher G, Gyssens M, Leinders D, Van den Bussche J, Van Gucht D, Vansumeren S (2015) Similarity and bisimilarity notions appropriate for characterizing indistinguishability in fragments of the calculus of relations. J Logic Comput 25(3):549–580
- Fletcher G, Gyssens M, Paredaens J, Van Gucht D, Wu Y (2016) Structural characterizations of the navigational expressiveness of relation algebras on a tree. J Comput Syst Sci 82(2):229–259
- Message Passing Interface Forum (2015) MPI: a message passing interface standard, version 3.1. <https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf>
- Gao S, Anyanwu K (2013) PrefixSolve: efficiently solving multi-source multi-destination path queries on RDF graphs by sharing suffix computations. In: WWW, pp 423–434
- Gubichev A, Bedathur SJ, Seufert S (2013) Sparqlng Kleene: fast property paths in RDF-3X. In: GRADES, pp 14:1–14:7
- Gurajada S, Theobald M (2016) Distributed set reachability. In: SIGMOD, pp 1247–1261
- Gurajada S, Seufert S, Miliaraki I, Theobald M (2014) TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing. In: SIGMOD, pp 289–300
- Harbi R, Abdelaziz I, Kalnis P, Mamoulis N (2015) Evaluating sparql queries on massive RDF datasets. Proc VLDB Endow 8(12):1848–1851. <https://doi.org/10.14778/2824032.2824083>
- Harris S, Gibbins N (2003) 3store: efficient bulk RDF storage. In: PSSS, pp 1–15
- Harris S, Lamb N, Shadbolt N (2009) 4store: the design and implementation of a clustered RDF store. In: SSWS
- Hellings J, Fletcher G, Havercort H (2012) Efficient external-memory bisimulation on DAGs. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2012, Scottsdale, 20–24 May 2012, pp 553–564
- Huang J, Abadi DJ, Ren K (2011) Scalable SPARQL querying of large RDF graphs. PVLDB 4(11): 1123–1134
- Lin J, Dyer C (2010) Data-intensive text processing with MapReduce. Synthesis lectures on human language technologies. Morgan & Claypool Publishers, San Rafael
- Luo Y, Picalausa F, Fletcher GHL, Hidders J, Vansumeren S (2012) Storing and indexing massive RDF

- datasets. In: Virgilio RD, Guerra F, Velegrakis Y (eds) Semantic search over the web. Springer, Berlin/Heidelberg, pp 31–60. https://doi.org/10.1007/978-3-642-25008-8_2
- Luo Y, de Lange Y, Fletcher G, De Bra P, Hidders J, Wu Y (2013a) Bisimulation reduction of big graphs on MapReduce. In: Big data – proceedings of 29th British National conference on databases, BNCOD 2013, Oxford, 8–10 July 2013, pp 189–203
- Luo Y, Fletcher G, Hidders J, Wu Y, De Bra P (2013b) External memory k-bisimulation reduction of big graphs. In: 22nd ACM international conference on information and knowledge management, CIKM'13, San Francisco, 27 Oct–1 Nov 2013, pp 919–928
- Maccioni A, Abadi DJ (2016) Scalable pattern matching over compressed graphs via dedensification. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, 13–17 Aug 2016, pp 1755–1764
- Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: SIGMOD, pp 135–146
- Maneth S, Peternek F (2016) Compressing graphs by grammars. In: 32nd IEEE international conference on data engineering, ICDE 2016, Helsinki, 16–20 May 2016, pp 109–120
- Milo T, Suciu D (1999) Index structures for path expressions. In: Database theory – ICDT'99, proceedings of 7th international conference, Jerusalem, 10–12 Jan 1999, pp 277–295
- Neumann T, Weikum G (2010a) The RDF-3X engine for scalable management of RDF data. VLDB J 19(1): 91–113
- Neumann T, Weikum G (2010b) x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. PVLDB 3:256–263
- Peng P, Zou L, Özsu MT, Chen L, Zhao D (2016) Processing sparql queries over distributed RDF graphs. VLDB J 25(2):243–268. <https://doi.org/10.1007/s00778-015-0415-0>
- Picalausa F, Luo Y, Fletcher G, Hidders J, Vansumeren S (2012) A structural approach to indexing triples. In: The semantic web: research and applications – proceedings of 9th extended semantic web conference, ESWC 2012, Heraklion, 27–31 May 2012, pp 406–421
- Picalausa F, Fletcher G, Hidders J, Vansumeren S (2014) Principles of guarded structural indexing. In: Proceedings of 17th international conference on database theory (ICDT), Athens, 24–28 Mar 2014, pp 245–256
- Potter A, Motik B, Nenov Y, Horrocks I (2016) Distributed RDF query answering with dynamic data exchange. In: ISWC, pp 480–497. https://doi.org/10.1007/978-3-319-46523-4_29
- Przyjaciel-Zablocki M, Schätzle A, Hornung T, Lausen G (2011) RDFPath: path query processing on large RDF graphs with MapReduce. In: ESWC, pp 50–64
- RDF (2014) Resource description framework. <http://www.w3.org/RDF/>
- Rohloff K, Schantz RE (2011) Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store. In: DIDC, pp 35–44
- Sarwat M, Elnikety S, He Y, Mokbel MF (2013) Horton+: a distributed system for processing declarative reachability queries over partitioned graphs. PVLDB 6(14):1918–1929
- Seufert S, Anand A, Bedathur SJ, Weikum G (2013) FERRARI: flexible and efficient reachability range assignment for graph indexing. In: ICDE, pp 1009–1020
- Sidiropoulos L, Goncalves R, Kersten M, Nes N, Manegold S (2008) Column-store support for RDF data management: not all swans are white. PVLDB 1(2):1553–1563
- SPA (2013) SPARQL 1.1 overview. <https://www.w3.org/TR/sparql11-overview/>
- Su J, Zhu Q, Wei H, Yu JX (2017) Reachability querying: can it be even faster? IEEE Trans Knowl Data Eng 29(3):683–697
- Then M, Kaufmann M, Chirigati F, Hoang-Vu T, Pham K, Kemper A, Neumann T, Vo HT (2014) The more the merrier: efficient multi-source graph traversal. PVLDB 8(4):449–460
- Tian Y, Balmin A, Corsten SA, Tatikonda S, McPherson J (2013) From “think like a vertex” to “think like a graph”. PVLDB 7(3):193–204. <http://www.vldb.org/pvldb/vol7/p193-tian.pdf>
- Tsialiamanis P, Sidiropoulos L, Fundulaki I, Christophides V, Boncz PA (2012) Heuristics-based query optimisation for SPARQL. In: EDBT, pp 324–335
- Webber J (2012) A programmatic introduction to Neo4j. In: SPLASH, pp 217–218
- Weiss C, Karras P, Bernstein A (2008) Hexastore: sextuple indexing for semantic web data management. PVLDB 1(1):1008–1019
- Xin RS, Gonzalez JE, Franklin MJ, Stoica I (2013) GraphX: a resilient distributed graph system on Spark. In: GRADES
- Yuan P, Liu P, Wu B, Jin H, Zhang W, Liu L (2013) TripleBit: a fast and compact system for large scale RDF data. PVLDB 6(7):517–528
- Zaharia M, Chowdhury NMM, Franklin M, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. Technical report UCB/EECS-2010-53, EECS Department, UC Berkeley, <http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-53.html>
- Zeng K, Yang J, Wang H, Shao B, Wang Z (2013) A distributed graph engine for web scale RDF data. PVLDB 6(4):265–276
- Zhang X, Chen L, Tong Y, Wang M (2013) EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud. In: ICDE, pp 565–576
- Zou L, Mo J, Chen L, Özsu MT, Zhao D (2011) gStore: answering SPARQL queries via subgraph matching. PVLDB 4(8):482–493

Industry 4.0

- ▶ Big Data Warehouses for Smart Industries

Inference

- ▶ Automated Reasoning
- ▶ Reasoning at Scale

Influence Analytics in Graphs

Yuichi Yoshida¹, Panayiotis Tsaparas², and Laks V. S. Lakshmanan³

¹National Institute of Informatics, Tokyo, Japan

²Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

³The University of British Columbia, Vancouver, BC, Canada

Definitions

“Influence analytics in graphs” is the study of the dynamics of influence propagation over a network. Influence includes anything that can be transferred or shaped through network interactions, such as pieces of information, actions, behavior, or opinions. Influence analytics includes modeling of the process of influence propagation and addressing algorithmic challenges that arise in solving optimization problems related to the dynamic propagation process.

Overview

Diffusion process on networks is one of the most important dynamics studied in network analysis, which can represent social influence such as the

word-of-mouth effect of a rumor in a social network, the spread of an infectious disease, opinion formation through social interaction, and many other phenomena that could happen online or offline. Although diffusion process on networks has been studied from the 1960s, in various research areas including sociology and economy, its importance has grown considerably because of the rapid growth of the web graph and online social networks. Socially, it has become more important because more and more influence is exerted on people through those networks. Academically, it has become more important because the massive scale of the networks raises new algorithmic challenges.

Among the various problems defined on the network diffusion process, this article focuses on an optimization perspective, that is, finding a small seed set of nodes in a given network that maximizes the total spread of influence in the final or the convergent state of the diffusion process. In particular, as notable examples, influence maximization arising in viral marketing and opinion maximization arising in opinion formation are discussed.

Influence Propagation

A social network acts as a medium for the spread of information, ideas, and innovations among its members. The idea of the *influence maximization problem*, which arose in the context of viral marketing, is controlling the spread so that an idea or product is widely adopted through word-of-mouth effects. More specifically, by targeting a small number of influential individuals, say, giving them free samples of the product, the goal is to trigger a cascade of influence by which friends will recommend the product to other friends, and many individuals will ultimately try it. One can rephrase this problem using graph terminology, that is, the goal is to find a set of small number of nodes, called a *seed set*, in a given network that maximizes the number of influenced nodes in the end of an influence propagation process.

Models

Most influence propagation models studied in the influence maximization problem consider a propagation process over a directed network $G = (V, E)$ that unfolds in discrete time steps in which each node is active (i.e., adopts an idea or innovation) or inactive. As representative models, this article focuses on the independent cascade (IC) model and the linear threshold (LT) model.

For a node $v \in V$, let $N_v^- = \{u \in V : uv \in E\}$ and $N_v^+ = \{w \in V : vw \in E\}$, denote the set of in-neighbors and out-neighbors of v , respectively.

Independent Cascade Model

The *independent cascade model* was firstly studied by Goldenberg et al. (2001) in the context of marketing. In this model, each arc $e \in E$ has a propagation probability $p_e \in [0, 1]$. The propagation process unfolds in discrete steps according to the following randomized rule:

- In step 0, some nodes are activated.
- When a node $v \in V$ becomes active in step $t \geq 0$, it is given a single chance of activating each inactive out-neighbor $w \in N_v^+$, which succeeds with probability p_{vw} . If v succeeds, then w will become active in step $t + 1$.
- The process terminates when no more activation is possible.

Note that, whether or not v succeeds in activating w , it cannot make any further attempts to activate w in subsequent steps.

An example of a propagation process in the IC model is shown in Fig. 1.

Linear Threshold Model

The *linear threshold model* was introduced by Granovetter (1978) to model collective behavior such as the riot and residential segregation. The threshold denotes the proportion

of others who must make a particular decision before an individual does so.

In the LT model, each arc $e \in E$ has a nonnegative weight b_e such that $\sum_{uv \in E} b_{uv} \leq 1$ for every $v \in V$. In this model, a node $v \in V$ is influenced by each in-neighbor $u \in N_v^-$ according to the weight b_{uv} . If the total amount of influence exceeds a randomly chosen threshold $\theta_v \in [0, 1]$, then v is activated. More formally, the propagation process unfolds in discrete steps according to the following rule.

- In step 0, some nodes are activated and each node $v \in V$ chooses a threshold $\theta_v \in [0, 1]$ uniformly at random from the interval $[0, 1]$.
- In step $t \geq 1$, all nodes that were active in step $t - 1$ remain active, and a node $v \in V$ is activated if the total weight of its active in-neighbors is at least θ_v .
- The process terminates when no more activation is possible.

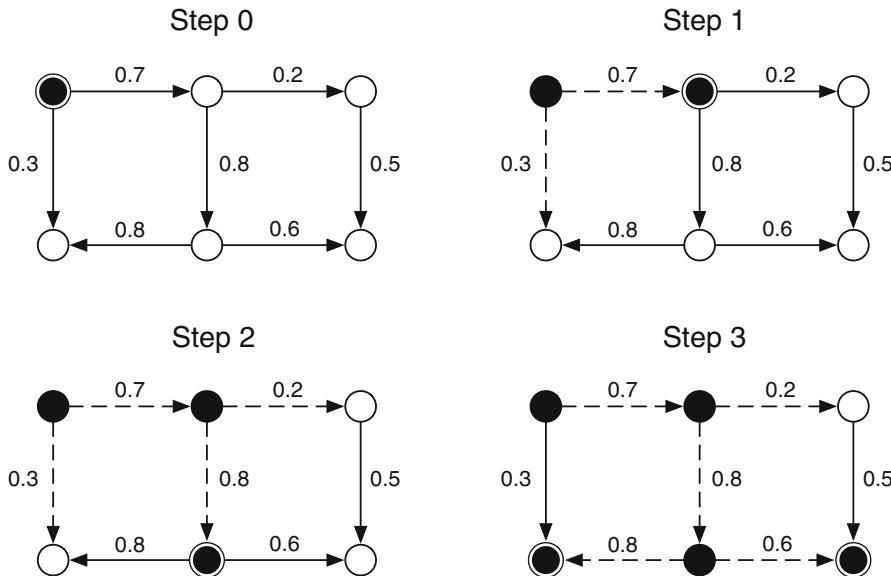
The fact that the thresholds are randomly selected is intended to model the lack of knowledge of their values. Note that, after the thresholds are fixed, the propagation process is deterministic.

An example of a propagation process in the LT model is shown in Fig. 2.

Influence Maximization

For the IC and LT models, one can formalize the influence maximization problem as follows: First, let $f : 2^V \rightarrow \mathbb{R}$ be a set function such that $f(S)$ is the expected number of activated nodes in the end of the propagation process when the propagation process starts with an initial seed set $S \subseteq V$. Then, given an additional integer parameter $1 \leq k \leq |V|$, the goal is to maximize $f(S)$ subject to $|S| \leq k$.

As computing the optimal solution is NP-hard, the following greedy algorithm is frequently used:



Influence Analytics in Graphs, Fig. 1 Example of a propagation process in the independent cascade model. Black nodes, black nodes with white boundary, and white nodes are active, activated in the current step, and inactive, respectively. Real numbers on arcs denote the success probability of activating the node in the head when the tail node is active. Step 0: The top-left vertex

is initially activated. Step 1: The top-middle vertex is activated, whereas the bottom-left vertex remains inactive. The arcs once used in the activation process will be no longer used (dashed arcs). Step 2: The bottom-middle vertex is activated. Step 3: The bottom-left and bottom-right vertices are activated

- Let S be an empty set.
- For k times:
 - Let $v \in V \setminus S$ be the vertex such that it maximizes $f(S \cup \{v\}) - f(S)$ and $f(S \cup \{v\}) - f(S) > 0$ (break ties arbitrarily).
 - Add v to S .
- Return S .

The greedy algorithm above is not merely a heuristic; it achieves $(1 - 1/e)$ -approximation to the optimal solution (Nemhauser and Wolsey 1978) for the IC and LT models due to the following key properties: (i) *monotonicity*, i.e.,

$$f(S) \leq f(T)$$

for every $S \subseteq T \subseteq V$, and (ii) *submodularity*, i.e.,

$$f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$$

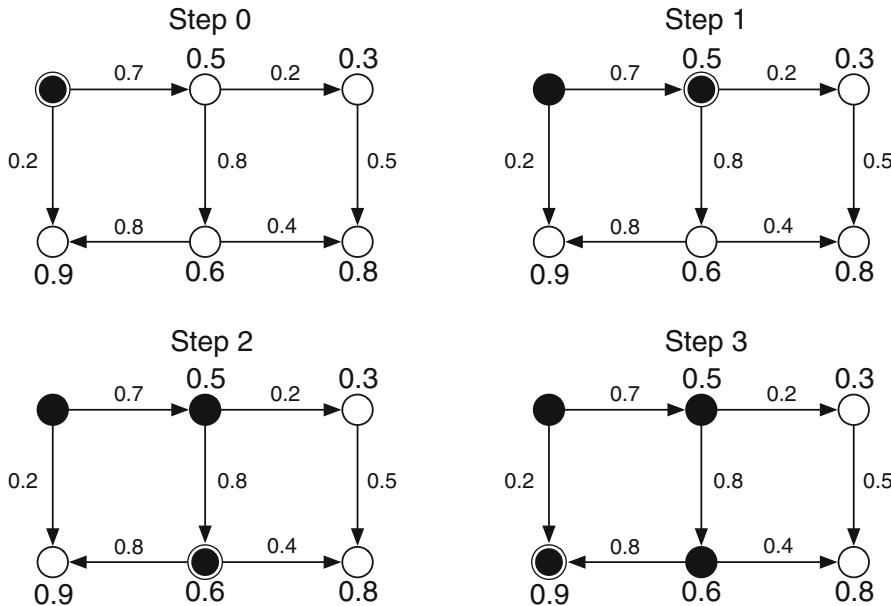
for every $S, T \subseteq V$. Monotonicity is equivalent to

$$f(S) \leq f(S \cup \{v\})$$

for every $S \subseteq V$ and $v \in V \setminus S$, meaning that the expected number of activated nodes is non-decreasing as a new node v is added into a seed set S . Submodularity is equivalent to

$$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$$

for every $S \subseteq T \subseteq V$ and $v \in V \setminus T$, meaning that the increase in the expected number of activated nodes from adding a new node v into a seed set S is no smaller than that into seed set



Influence Analytics in Graphs, Fig. 2 Example of a propagation process in the linear threshold model. Black nodes, black nodes with white boundary, and white nodes are active, activated in the current step, and inactive, respectively. Real numbers on arcs denote their weights, and

ones on nodes denote their thresholds. The top-left vertex is initially activated in Step 0 and then the top-middle, bottom-middle, and bottom-left vertices are activated in Steps 1, 2, and 3, respectively

$T \supseteq S$. Both properties are intuitively true and formally proved by Kempe et al. (2003).

An additional source of complexity in influence maximization is that evaluating the value of $f(S)$ is a $\#P$ -hard problem because $f(S)$ is defined as the expectation over exponentially many realizations of the propagation process. For details, Chen et al. (2013). Fortunately, the value of $f(S)$ can be well approximated by simulating the propagation process several times and taking the average of the number of activated nodes across the runs. Those approximated values are sufficient to obtain $(1 - 1/e - \varepsilon)$ -approximation in polynomial time for any constant $\varepsilon > 0$ (Kempe et al. 2003). This approach does not scale to large networks.

Borgs et al. (2014) achieved a scalable approximation algorithm for the influence maximization problem under the IC and LT models that runs in almost linear time in the network size. The current best time complexity for both models is $O(k(n + m) \log n / \varepsilon^2)$ and is due to Tang et al. (2014).

Learning the Model

An important question is where the edge probabilities (for the IC model) and edge weights (for the LT model) come from. These parameters are not readily available and thus must be learned. One approach that was pioneered by Saito et al. (2008) (for the IC model) and Goyal et al. (2010) (for the generalized threshold model) is to learn these parameters from action logs. An *action log* is a table $\text{Log}(\text{User}, \text{Action}, \text{Time})$ that records for every user and action performed by the user, the time at which it was performed.

Saito et al. (2008) assumed that the time stamps in an actual log of actions are partitioned into bins corresponding to discrete time steps. Given this, they develop an expectation maximization approach for learning the edge probabilities. The key idea is that based on the observed actions and the given network, one can formulate the likelihood of observing those actions given model parameters and learn the parameters as those that maximize the logarithm

of the said likelihood. A subtle point is that there may be many different ways to partition an actual action log into bins, and different partitions may lead to different values for the likelihood. The question of finding the “optimal” partitioning is not addressed by them. One inherent challenge with this approach is that the algorithm can take considerable time for learning model parameters on large networks and action logs.

Goyal et al. (2010) followed a simple frequentist approach and used a maximum likelihood estimator (MLE) for learning the edge weights under the generalized threshold model, which is a generalization of the LT model. In addition to a framework for learning model parameters, they show that their framework can handle the case where influence probabilities change (specifically, decay) over time (Indeed, in practice, one can expect influence to decay over time.). Using this, they show that in addition to predicting user activation, they can also predict the most likely time of activation of users. They also propose notions of user and action influenceability and ways of measuring them from an action log and show that for users with higher influenceability, both activation prediction and prediction of the most likely time of activation can be done more accurately. The learning algorithm proposed in Goyal et al. (2010) scales to large networks and action logs: the algorithm makes just one pass over the action log.

There has been significant subsequent work on learning influence models from given action propagation data, including improvements and extensions, which are omitted for lack of space.

Opinion Formation

Different from influence propagation models such as the IC and LT models, where nodes are in a binary state of being active or inactive, in opinion formation models, the state of a node $v \in V$ is a real value z_v which captures the *opinion* of the node. Typically, the opinion values range over the interval $[0, 1]$ (sometimes, $[-1, 1]$), where $z_v = 1$ signifies a strong positive opinion, while $z_v = 0$ signifies a strong negative opinion.

Similar to the influence propagation models, the underlying principle behind opinion formation models is that the state (opinion) of a node is determined by the states of the nodes in the neighborhood. Following the intuition that the opinions of individuals in real life are influenced from the opinions in their social circle, the opinion formation models assume a propagation of opinions in the network. However instead of activating the nodes, the opinion formation process adjusts the nodes’ opinions according to a specific model.

A variety of opinion formation processes have been proposed that aim to model different aspects of opinion dynamics (Degroot 1974; Hegselmann and Krause 2002; Castellano et al. 2009), such as, consensus (convergence to a single opinion), polarization (clustering of opinions around two extreme positions), or plurality (existence of multiple clusters of opinions). This article focuses on algorithmic challenges that arise in these processes. More specifically, it focuses on the problem of *opinion maximization*, where the goal is to maximize (say) the positive opinion in the network.

The Friedkin and Johnsen Model

A widely adopted opinion formation model is the model of Friedkin and Johnsen (1990). In this model, the network is undirected, and every node $v \in V$ has an *internal* opinion value $s_v \in [0, 1]$ and an *expressed* opinion value $z_v \in [0, 1]$. The internal opinion corresponds to the prior beliefs of the individual, and it is fixed and unchanged throughout the opinion formation process. The expressed opinion is malleable, and its value is the result of the influences exerted on the node by its neighbors in the graph.

Mathematically, the expressed opinion is defined as follows:

$$z_v = \frac{s_v + \sum_{u \in N_v} z_u}{1 + |N_v|}, \quad (1)$$

where $N_v = \{w \in V : vw \in E\}$ is the set of neighbors of v . Equation (1) intuitively means that the expressed opinion of a node is determined by taking average over its own internal opinion and the expressed opinions of its neighbors.

The expressed opinions can be computed by solving a system of linear equations. Let \mathbf{s} be the vector of internal opinions and \mathbf{z} the vector of expressed opinions. Then

$$\mathbf{z} = (\mathbf{I} + \mathbf{L})^{-1} \mathbf{s} \quad (2)$$

where \mathbf{L} is the Laplacian matrix of the adjacency matrix of the social graph.

The expressed opinions can also be viewed as the output of a dynamic system which iteratively updates the opinion values. The opinion formation model proceeds as follows:

- In step 0, initialize for each node $v \in V$, $z_v^0 = s_v$.
 - In step $t \geq 1$, all nodes update their expressed opinions as follows:
- $$z_v^t = \frac{s_v + \sum_{u \in N_v} z_u^{t-1}}{1 + |N_v|}$$
- Repeat the process until all expressed opinion values converge.

There are several interpretations of the Friedkin and Johnsen model. Bindel et al. (2015) showed that the opinion values are the result of the selfish behavior of nodes, where a node v tries to minimize their personal cost of adopting opinion value z_v , defined as

$$C(z_v) = (z_v - s_v)^2 + \sum_{u \in N_v} (z_v - z_u)^2$$

The first term in the cost function is the cost incurred to node v for compromising their own internal beliefs. The second term is the cost incurred to node v for nonconforming with the opinions of their social neighborhood.

A different interpretation is offered by Gionis et al. (2013), where they model the opinion formation process as a random walk on a graph with absorbing nodes. Intuitively, the social network

is viewed as an electrical network, where edges are resistances. Each node is connected to its own voltage source, with voltage equal to the internal opinion value of the node (in the absorbing random walk, these are absorbing nodes). The voltage value at each node is equal to the expressed opinion of the node.

Opinion Maximization

In the *opinion maximization* problem (Gionis et al. 2013), the goal is to maximize the sum of the opinion values in the network. This sum captures the average opinion in the network, which should be as favorable as possible. There are two ways to affect the average opinion in the network:

- I. Convince nodes to change their internal opinions to be as positive as possible (set $s_v = 1$). In the electrical network interpretation, this corresponds to setting the voltage of the power source of v to +1
- II. Convince nodes to adopt a positive expressed opinion (set $z_v = 1$). In this case the expressed opinion value remains fixed, and it is not iteratively updated. In the electrical network interpretation, this corresponds to turning node v into a voltage source with voltage +1.

The goal is to find a small set of nodes $S \subseteq V$ such that changing their opinions maximizes the sum of expressed opinion values. More formally, let G denote the social network. Let $g(\mathbf{z}; G, \mathbf{s}) = \sum_{v \in V} z_v$ be the sum of opinion values, and let $g(\mathbf{z}; S, G, \mathbf{s})$ be the sum of opinion values after changing the opinions of the nodes in S , by setting either their internal or expressed opinion to 1. The goal is to find a set S of k nodes so as to maximize $g(\mathbf{z}; S, G, \mathbf{s})$.

In Case I, the problem can be solved optimally in polynomial time by simply selecting S to be the nodes with the lowest internal opinion value. This follows from Eq.(2) and the fact that $g(\mathbf{z}; S, G, \mathbf{s}) = \sum_{v \in V} s_v$ (Gionis et al. 2013).

In Case II, the problem is NP-hard. Similar to influence maximization, the value of g as a

function of the set S is monotone and submodular, and thus, the greedy algorithm provides a $(1 - 1/e)$ -approximation to the optimal solution.

Future Directions

Although influence maximization and opinion maximization have been intensively studied, there are still many challenges that should be addressed.

The greedy method achieves $(1 - 1/e)$ -approximation to the optimal solution both for the influence maximization and opinion maximization problems. However, empirical performance seems to be much better than this theoretical guarantee and this gap should be filled.

The main focus of the influence maximization problem is to maximize the *expected* number of activated nodes. However, maximizing the number of activated nodes in less propagated scenarios is also important. To address this problem, Ohsaka and Yoshida (2017) exploited the notion of *conditional value at risk* (CVaR) from the field of financial risk measurement, which is the expected number of activated nodes in the worst α -fraction of the scenarios, where $\alpha \in (0, 1)$ is a parameter specified by a user. They proposed a polynomial-time algorithm that produces a portfolio over initial seed sets with a provable guarantee on its CVaR. Optimizing other measures on probability distributions such as variance will be interesting.

Information propagation models require some parameters on arcs (edge probabilities or edge weights), which are typically unknown for real networks. As previously discussed, approaches for learning these parameters have been developed. However, there may be noise in the learned parameters, on account of the action log being noisy or because of the inherent uncertainty of the learning problem. Hence, it is natural to introduce some intervals on those parameters and seek an initial seed set that has a good approximation guarantee no matter how parameters are chosen from the associated intervals. This problem is called the *robust influence maximization*, and

some positive and negative results are obtained by Chen et al. (2016) and He and Kempe (2016). Tightening the gap between them is an interesting problem.

Influence and opinion maximization can be considered together, in a model where nodes carry an opinion value, and the goal is to maximize the overall opinion of the activated nodes (e.g., see the work of Zhang et al. 2013). This formulation opens the possibility of combining techniques from both problems to design better models and algorithms.

The idea of selecting a few nodes to affect node opinions has also been applied to tasks other than opinion maximization. For example, Matakos et al. (2017) applied this idea for reducing polarization, while Golshan and Terzi (2017) applied it for minimizing tension in a related model. It would be interesting to consider other applications of this idea in different settings.

Cross-References

- ▶ [Big Data Analysis for Social Good](#)
- ▶ [Big Data in Social Networks](#)
- ▶ [Link Analytics in Graphs](#)

References

- Bindel D, Kleinberg JM, Oren S (2015) How bad is forming your own opinion? *Games Econ Behav* 92:248–265
- Borgs C, Brautbar M, Chayes J, Lucier B (2014) Maximizing social influence in nearly optimal time. In: Proceedings of the 25th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 946–957
- Castellano C, Fortunato S, Loreto V (2009) Statistical physics of social dynamics. *Rev Mod Phys* 81(2):591–646
- Chen W, Lakshmanan LVS, Castillo C (2013) Information and influence propagation in social networks. Synthesis lectures on data management. Morgan & Claypool Publishers, San Rafael
- Chen W, Lin T, Tan Z, Zhao M, Zhou X (2016) Robust influence maximization. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 795–804
- Degroot MH (1974) Reaching a consensus. *J Am Stat Assoc* 69(345):118–121

- Friedkin NE, Johnsen EC (1990) Social influence and opinions. *J Math Sociol* 15(3–4): 193–206
- Gionis A, Terzi E, Tsaparas P (2013) Opinion maximization in social networks. In: Proceedings of the 13th SIAM international conference on data mining (SDM), pp 387–395
- Goldenberg J, Libai B, Muller E (2001) Talk of the network: a complex systems look at the underlying process of word-of-mouth. *Mark Lett* 12(3): 211–223
- Golshan B, Terzi E (2017) Minimizing tension in teams. In: Proceedings of the 26th ACM international conference on information and knowledge management (CIKM), pp 1707–1715
- Goyal A, Bonchi F, Lakshmanan LV (2010) Learning influence probabilities in social networks. In: Proceedings of the 3rd ACM international conference on web search and data mining (WSDM), pp 241–250
- Granovetter M (1978) Threshold models of collective behavior. *Am J Sociol* 83(6):1420–1443
- He X, Kempe D (2016) Robust influence maximization. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 885–894
- Hegselmann R, Krause U (2002) Opinion dynamics and bounded confidence models, analysis and simulation. *J Artif Soc Soc Simul* 5(3):1–33
- Kempe D, Kleinberg J, Tardos É (2003) Maximizing the spread of influence through a social network. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 137–146
- Matakos A, Terzi E, Tsaparas P (2017) Measuring and moderating opinion polarization in social networks. *Data Min Knowl Disc* 31(5):1480–1505
- Nemhauser GL, Wolsey LA (1978) Best algorithms for approximating the maximum of a submodular set function. *Math Oper Res* 3(3):177–188
- Ohsaka N, Yoshida Y (2017) Portfolio optimization for influence spread. In: Proceedings of the 26th international conference on world wide web (WWW), pp 977–985
- Saito K, Nakano R, Kimura M (2008) Prediction of information diffusion probabilities for independent cascade model. In: Proceedings of the 12th international conference on knowledge-based intelligent information and engineering systems (KES), pp 67–75
- Tang Y, Xiao X, Shi Y (2014) Influence maximization: near-optimal time complexity meets practical efficiency. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data (SIGMOD), pp 75–86
- Zhang H, Dinh TN, Thai MT (2013) Maximizing the spread of positive influence in online social networks. In: Proceedings of the 33rd IEEE international conference on distributed computing systems (ICDCS), pp 317–326

Information Filtering

- Big Data and Recommendation

Information Flows

- Definition of Data Streams

Information Integration

- Data Integration

Information Visualization

- Big Data Visualization Tools

In-Memory Transactions

Spyros Blanas

Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Definitions

In-memory transactions are database transactions that operate on data that are stored in memory. Because the main memory is much faster than a disk, the processing throughput of in-memory transactions can be orders of magnitude higher than when transactions manipulate data on disk. To realize this potential, an in-memory transaction processing engine needs to avoid contention, blocking, and context switching when accessing memory for high performance while guaranteeing the atomicity, isolation, and durability of in-memory transactions for correctness.

Overview

Database management systems were designed assuming that the database resides on disk, which is orders of magnitude slower than memory. Many design decisions about data storage and transaction management focused on improving the concurrency and the performance of disk accesses. Less attention was devoted in making accesses to memory efficient. Through the decades, however, the memory capacity of each server grew substantially as memory modules became denser and cheaper. As a consequence, the actively-updated part of many transactional databases can now be stored entirely in memory. This has profound implications for the architecture of database management systems for transaction processing workloads.

Modern in-memory database engines strive to eliminate contention, blocking, and context switching when accessing memory-resident data. Through a combination of novel concurrency control algorithms, clever in-memory data storage methods, and efficient implementations, an in-memory transaction processing engine can process millions of transactions per second on a single server.

The impressive performance gains from in-memory transaction processing have attracted the interest of both established database vendors and disruptors to offer in-memory transaction processing capabilities in commercial products. Exemplars of established database management systems that are optimized for in-memory transactions include Microsoft SQL Server (Hekaton), SAP HANA, and Oracle 12c. Many start-ups, including VoltDB, Aerospike, and MemSQL, also offer products with in-memory transaction capabilities.

Key Research Findings

The intellectual underpinning of in-memory transaction processing is research on how to improve the concurrency of non-conflicting operations in memory. One research direction

focuses on in-memory data structures for highly concurrent accesses, frequently using lock-free techniques to eliminate blocking and context switching. Another research direction is fast concurrency control algorithms that enforce transactional atomicity and isolation by delaying or aborting transactions with conflicting logical operations. In addition, an in-memory transaction processing system needs to ameliorate contention at the tail of the transaction log and quickly recover the database after a crash. The research challenge is how to efficiently execute thousands of very short in-memory transactions concurrently.

Data Storage

Database systems that are optimized for in-memory transactions do not organize data in a traditional buffer pool. Prior work points to a number of disadvantages of using a buffer pool for in-memory data management. A buffer pool tracks pages, and updating pages requires costly coordination using locks (Harizopoulos et al. 2008). Lock-based synchronization blocks the thread that issues a conflicting request, which needs to either context switch or spin until the lock is granted. The overhead of context switching and spinning can be thousands of CPU cycles, which is comparable to the compute cost of completing an in-memory transaction. Therefore, conflicts on page locks of frequently accessed tuples would significantly underutilize a modern CPU for in-memory transaction processing. In addition, updating data in pages causes spurious conflicts: non-conflicting requests that access different tuples in the same page may spuriously conflict on the page lock.

Instead, database systems that are optimized for in-memory transactions avoid locking when reading data. Research has focused on two families of data structures: hash-based data structures use a hash function to determine the access location; tree-based data structures organize data hierarchically. In general, hash-based data structures are faster when looking up a single key as they do not need to traverse a hierarchy of objects. Tree-based structures out-

perform when subsequent accesses are sequential because they benefit from spatial and temporal locality.

Hash-Based Indexing

Hash-based indexing has been proposed for data storage in main memory transaction processing systems such as Hekaton (Diaconu et al. 2013). Multiple attributes can be indexed; each indexable attribute will have a separate in-memory hash table structure. An in-memory hash table is created by hashing the attribute of every tuple in a table. Every hash bucket is the root of a singly linked list that stores all the tuples that hash to the same value. By tracking individual tuples, this design avoids spurious conflicts during concurrent operations. In addition, using a singly linked list allows for lock-free operations on the hash table. Readers start their search at the hash table bucket and follow pointers to the next data item. Modifying the linked list uses atomic compare-and-swap (CAS) instructions.

Avoiding lock contention and blocking, however, requires lock-free memory allocation and reclamation. A lock-free memory allocator guarantees progress regardless of whether some threads are delayed or even killed and regardless of scheduling policies. Prior work has investigated how memory allocators can offer freedom from deadlocks, gracefully handle asynchronous signals, safely handle priority inversion, and tolerate thread preemption and thread termination (Michael 2004b). Another challenge is safe memory reclamation. In particular, the problem is how to allow memory to be freed in a lock-free manner that guarantees that no thread currently accesses this memory. One solution from prior work keeps the pointers that a thread accesses in a globally visible structure and carefully monitors when a reclamation is safe (Michael 2004a).

Bw-Tree

A tree-based in-memory data structure is the Bw-Tree (Levandoski et al. 2013). The Bw-Tree organizes data in pages and closely resembles the Blink-tree data structure (Lehman and Yao 1981). Nodes in the Bw-Tree contain key and

pointer pairs, sorted by key, to direct searches to lower levels of the Bw-Tree. In addition, all nodes contain a side link pointer that points to the immediate right sibling on the same level of the tree. The Bw-Tree thus offers logarithmic access time for key lookups and linear access time for queries on a one-dimensional key range. Range queries are particularly efficient as they access contiguous data items. Unlike disk-based structures that impose a pre-defined page size, pages in a Bw-tree are elastic: they can grow to arbitrarily large sizes until split by a system management operation.

The main distinction from disk-oriented tree structures is that all page references in the Bw-Tree are logical references to a page identifier. This level of indirection allows readers to access the data structure in a lock-free manner. An in-memory hash table, called the mapping table, maintains the current mapping of page identifiers to physical memory locations. The Bw-Tree never updates in place (i.e., it never modifies the contents of a page). Instead, updates in a Bw-Tree create delta records that are prepended to existing pages and describe the update. The memory address of the delta record is atomically installed in the mapping table using the atomic compare-and-swap (CAS) instruction. Hence, a concurrent read request that looks up the physical location of a logical page identifier in the mapping table will either see the new page address (that contains the delta record) or the old page address (that skips the delta record), but it will never see an inconsistent state.

Masstree

Another tree-based in-memory data structure is Masstree (Mao et al. 2012). Masstree combines properties of a trie and a B⁺-tree in one data structure. The Silo transaction processing engine uses a structure similar to Masstree for data storage (Tu et al. 2013). Masstree tracks individual tuples and carefully coordinates writes for high concurrency.

Masstree consists of multiple trees arranged in layers. Each layer is indexed by a different 8-byte slice of the key. Therefore, layer 0 has a single tree that is indexed by bytes 0–7 of the

key; trees in layer 1, the next deeper layer, are indexed by bytes 8–15; trees in layer 2 by bytes 16–23; and so forth. Each tree contains interior nodes and border (leaf) nodes. Interior nodes form the trie, whereas border nodes store values (akin to a B^+ -tree) or pointers to trees in deeper layers. Masstree creates additional layers as needed when an insertion cannot use an existing tree.

Masstree is designed for highly concurrent accesses. Instead of a lock, each node in the Masstree structure has a version counter. A writer increments the version counter of a node to denote an in-progress update and increments the version number again after the update has been installed. Readers coordinate with writers by checking the version of a node before a read and comparing this number to the version after the operation. If the two version numbers differ or a version number is an odd number, the reader may have observed an inconsistent state and will retry. Therefore, conflicting requests in Masstree may restart, but readers will never write globally accessible shared memory.

Concurrency Control Algorithms

In-memory concurrency control algorithms guarantee the atomicity and isolation of in-memory transactions. The goal is serializability, that is, to allow transactions to execute concurrently while appearing to the user that they run in isolation. If the concurrency control algorithm detects an operation that violates serializability, such as two transactions updating the same item at the same time, it can delay or abort a transaction. The challenge is detecting and resolving any conflicts efficiently.

A widely used concurrency control protocol is single-version two-phase locking (Bernstein et al. 1987). In this protocol, transactions acquire read or write locks on a tuple, perform the operation, and release the lock at the end of the transaction. Lower isolation levels than serializable permit a transaction to release read locks earlier.

Single-version two-phase locking poses three problems for in-memory transactions. First, writers block readers in single-version two-phase locking. It is counterintuitive to many users why a

transaction that only reads should wait for update transactions, especially if the query does not need to access the latest data. In addition, blocking introduces the possibility of deadlocks where two transactions wait on each other. This requires the system to either perform expensive deadlock detection to abort a transaction that participates in a wait-for cycle or burden the user to specify a timeout parameter after which the transaction will be restarted. Finally, single-version two-phase locking triggers additional memory writes to the locks: a transaction needs to perform twice as many writes to locks (one write to acquire, one write to release) as the number of the records that it will access.

For these reasons, in-memory transaction processing research has looked for alternatives to single-version two-phase locking for in-memory transactions. Concurrency control algorithms for in-memory transactions can be classified into partitionable and non-partitionable categories.

Concurrency Control for Partitionable Workloads

Many transaction processing workloads exhibit a fairly predictable access pattern that allows a transaction processing system to partition the database such that most transactions only access a single partition. This access predictability is often because of the hierarchical structure of database entities, such as bank branches and accounts, customers and orders, or suppliers and parts.

The H-Store prototype (Kallman et al. 2008) abandons complex concurrency control protocols and just executes transactions serially on different database partitions. Multi-partition transactions, however, become more expensive as they require coordination across partitions, even if these partitions are assigned to different threads in the same node.

The DORA prototype (Pandis et al. 2010) proposes a shared-everything design that assigns data to threads. Transactions are decomposed into subtransactions that form a transaction flow graph. DORA then carefully routes subtransactions to the appropriate partition for execution to maintain serializability.

Partition-based approaches work very well if one can devise a good partitioning strategy. This may be challenging even for domain experts, however, as the query workload may change unpredictably. Automated methods can create a partitioning scheme for a given database and query workload. Schism (Curino et al. 2010) is a workload-aware database partitioner that attempts to minimize the number of distributed transactions while producing balanced partitions. Schism models this as a graph problem, where graph nodes are tuples and graph edges are co-accesses by the same transaction. Obtaining n balanced partitions that minimize inter-partition traffic requires performing n min-cuts to the graph. Sampling can be used to reduce the size of these graphs for very large databases. Clay (Serafini et al. 2016) proposes to perform partitioning in an adaptive and online fashion by carefully migrating tuple groups across partitions.

Concurrency Control for Non-partitionable Workloads

Not all in-memory transaction processing workloads are partitionable. Hekaton (Diaconu et al. 2013) and Silo (Tu et al. 2013) are designed for non-partitionable workloads and use optimistic multi-version concurrency control for transaction isolation. This concurrency control algorithm is rooted in the observation is that the short duration of in-memory transactions makes conflicts rare in a massive in-memory database. Optimistic concurrency control executes transactions speculatively until the end. Before making the writes of the transaction visible, optimistic concurrency control then verifies that the transaction did not observe or produce any inconsistent state because of conflicts.

Optimistic concurrency control validates that a transaction is serializable by comparing the set of accessed records of the committing transaction with the set of accessed records of every other active transaction in the system (Kung and Robinson 1981). This check becomes proportionally more expensive as the number active transactions increase in the system. This validation mechanism becomes a bottleneck with the massive

concurrency of modern multi-socket, multi-core CPUs for in-memory transactions.

The Hekaton in-memory concurrency control protocol (Larson et al. 2011) directly installs tentative versions in the database and carefully controls the visibility of these pending records by active transactions. A transaction then only needs to check once with the database to verify that no conflicting versions appeared or disappeared from the view. This permits the validation to be done in parallel from multiple threads without additional coordination. Compared with two-phase locking, non-conflicting transactions in the Hekaton in-memory concurrency control protocol perform an additional read per accessed record during validation, instead of two additional writes to acquire and release locks.

Multi-versioning allows readers to bypass writers. By retaining multiple versions, an in-memory transaction processing system can redirect readers to earlier versions instead of waiting for the writer to complete its transaction. Logically, the read transaction appears as if it arrived “just before” a write transaction. However, storing multiple versions also has drawbacks. First, the memory footprint of the database increases proportionally to the number of versions kept. Second, a garbage collection mechanism needs to determine when old versions will no longer be visible by any transaction and remove them from in-memory data structures. Recent work systematically explores the trade-offs between concurrency control protocol, version storage, garbage collection, and index management (Wu et al. 2017).

Durability Mechanisms for In-Memory Transactions

In-memory transaction processing systems must guarantee transaction durability in the presence of failures. The absence of a buffer pool means that “dirty” pages will never be written to durable storage. Therefore, in a main memory setting, traditional redo/undo logging and recovery as developed by ARIES (Mohan et al. 1992) is commonly replaced by redo-only logging.

Two challenges are ameliorating contention for the tail of the log and controlling the fre-

quency of checkpointing. Early lock release, consolidated buffer allocation, and asynchronous I/O move the latency of hardening log frames out of the critical path (Johnson et al. 2010). The checkpointing frequency is a trade-off between steady-state performance and recovery time: frequent checkpointing takes CPU cycles away from transaction processing, while checkpointing infrequently means substantial recovery work on failure.

Other Research Directions

In-memory transaction processing is a very active and evolving research field. Other research efforts improve the performance of in-memory transaction processing in the presence of heterogeneous workloads (Kim et al. 2016) and hotspots (Ren et al. 2016; Wang and Kimura 2016) or take advantage of determinism in transaction execution (Thomson et al. 2012). Another research direction considers properties of the serialization graph to enforce serializability (Wang et al. 2017) and reduce false aborts (Yuan et al. 2016). Multi-versioning has been shown to ameliorate lock contention through an indirection table (Sadoghi et al. 2014). Finally, new logging and recovery protocols promise to leverage byte-addressable non-volatile memory for in-memory transaction durability (Arulraj et al. 2016; Kimura 2015).

References

- Arulraj J, Perron M, Pavlo A (2016) Write-behind logging. *VLDB* 10(4):337–348
- Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Reading
- Curino C, Zhang Y, Jones EPC, Madden S (2010) Schism: a workload-driven approach to database replication and partitioning. *VLDB* 3(1):48–57
- Diaconu C, Freedman C, Ismert E, Larson PA, Mittal P, Stonecipher R, Verma N, Zwilling M (2013) Hekaton: SQL server’s memory-optimized OLTP engine. In: Proceedings of the 2013 ACM SIGMOD international conference on management of Data (SIGMOD’13). ACM, New York, pp 1243–1254. <https://doi.org/10.1145/2463676.2463710>
- Harizopoulos S, Abadi DJ, Madden S, Stonebraker M (2008) OLTP through the looking glass, and what we found there. In: SIGMOD’08: proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, New York, pp 981–992. <https://doi.org/10.1145/1376616.1376713>
- Johnson R, Pandis I, Stoica R, Athanassoulis M, Ailamaki A (2010) Aether: a scalable approach to logging. *Proc VLDB Endow* 3(1–2):681–692. <https://doi.org/10.14778/1920841.1920928>
- Kallman R, Kimura H, Natkins J, Pavlo A, Rasin A, Zdonik S, Jones EPC, Madden S, Stonebraker M, Zhang Y, Hugg J, Abadi DJ (2008) H-Store: a high-performance, distributed main memory transaction processing system. *Proc VLDB Endow* 1(2):1496–1499. <https://doi.org/10.1145/1454159.1454211>
- Kim K, Wang T, Johnson R, Pandis I (2016) Ermia: fast memory-optimized database system for heterogeneous workloads. In: Proceedings of the 2016 international conference on management of data (SIGMOD’16). ACM, New York, pp 1675–1687. <https://doi.org/10.1145/2882903.2882905>
- Kimura H (2015) Foedus: OLTP engine for a thousand cores and NVRAM. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data (SIGMOD’15). ACM, New York, pp 691–706. <https://doi.org/10.1145/2723372.2746480>
- Kung HT, Robinson JT (1981) On optimistic methods for concurrency control. *ACM Trans Database Syst* 6(2):213–226. <https://doi.org/10.1145/319566.319567>
- Larson P, Blanas S, Diaconu C, Freedman C, Patel JM, Zwilling M (2011) High-performance concurrency control mechanisms for main-memory databases. *VLDB* 5(4):298–309
- Lehman PL, Yao SB (1981) Efficient locking for concurrent operations on b-trees. *ACM Trans Database Syst* 6(4):650–670. <https://doi.org/10.1145/319628.319663>
- Levandoski JJ, Lomet DB, Sengupta S (2013) The bw-tree: a b-tree for new hardware platforms. In: 2013 IEEE 29th international conference on data engineering (ICDE), pp 302–313. <https://doi.org/10.1109/ICDE.2013.6544834>
- Mao Y, Kohler E, Morris RT (2012) Cache craftiness for fast multicore key-value storage. In: Proceedings of the 7th ACM European conference on computer systems (EuroSys’12). ACM, New York, pp 183–196. <https://doi.org/10.1145/2168836.2168855>
- Michael MM (2004a) Hazard pointers: safe memory reclamation for lock-free objects. *IEEE Trans Parallel Distrib Syst* 15(6):491–504. <https://doi.org/10.1109/TPDS.2004.8>
- Michael MM (2004b) Scalable lock-free dynamic memory allocation. *SIGPLAN Not* 39(6):35–46. <https://doi.org/10.1145/996893.996848>
- Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P (1992) Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans Database Syst* 17(1):94–162. <https://doi.org/10.1145/128765.128770>
- Pandis I, Johnson R, Hardavellas N, Ailamaki A (2010) Data-oriented transaction execution. *VLDB* 3(1):928–939

- Ren K, Faleiro JM, Abadi DJ (2016) Design principles for scaling multi-core OLTP under high contention. In: Proceedings of the 2016 international conference on management of data (SIGMOD'16). ACM, New York, pp 1583–1598. <https://doi.org/10.1145/2882903.2882958>
- Sadoghi M, Canim M, Bhattacharjee B, Nagel F, Ross KA (2014) Reducing database locking contention through multi-version concurrency. Proc VLDB Endow 7(13):1331–1342. <https://doi.org/10.14778/2733004.2733006>
- Serafini M, Taft R, Elmore AJ, Pavlo A, Aboulnaga A, Stonebraker M (2016) Clay: fine-grained adaptive partitioning for general database schemas. PVLDB 10(4):445–456
- Thomson A, Diamond T, Weng SC, Ren K, Shao P, Abadi DJ (2012) Calvin: fast distributed transactions for partitioned database systems. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data (SIGMOD'12). ACM, New York, pp 1–12. <https://doi.org/10.1145/2213836.2213838>
- Tu S, Zheng W, Kohler E, Liskov B, Madden S (2013) Speedy transactions in multicore in-memory databases. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles (SOSP'13). ACM, New York, pp 18–32. <https://doi.org/10.1145/2517349.2522713>
- Wang T, Kimura H (2016) Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. PVLDB 10(2):49–60
- Wang T, Johnson R, Fekete A, Pandis I (2017) Efficiently making (almost) any concurrency control mechanism serializable. VLDB J 26(4):537–562. <https://doi.org/10.1007/s00778-017-0463-8>
- Wu Y, Arulraj J, Lin J, Xian R, Pavlo A (2017) An empirical evaluation of in-memory multi-version concurrency control. PVLDB 10(7):781–792
- Yuan Y, Wang K, Lee R, Ding X, Xing J, Blanas S, Zhang X (2016) BCC: reducing false aborts in optimistic concurrency control with low cost for in-memory databases. PVLDB 9(6):504–515

Integration-Oriented Ontology

Sergi Nadal and Alberto Abelló
Polytechnic University of Catalonia, Barcelona,
Spain

Synonyms

[Ontology-based Data Access](#)

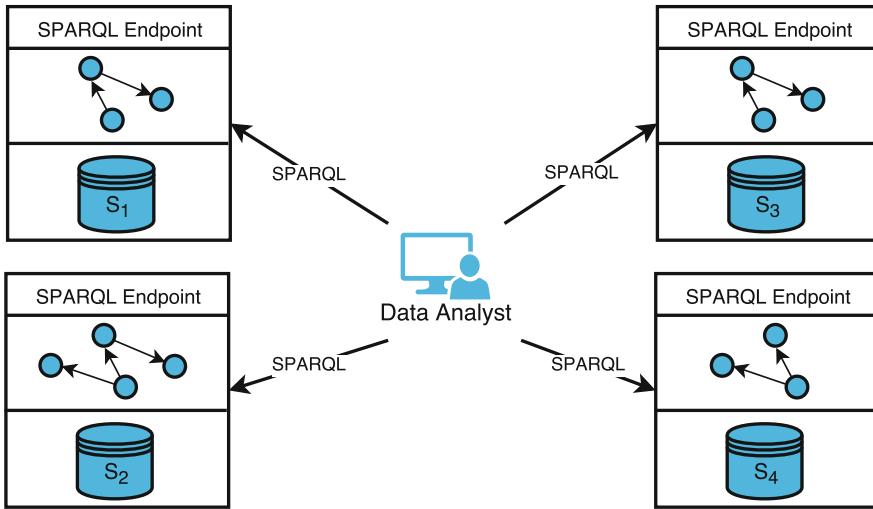
Definitions

The purpose of an integration-oriented ontology is to provide a conceptualization of a domain of interest for automating the data integration of an evolving and heterogeneous set of sources using Semantic Web technologies. It links domain concepts to each of the underlying data sources via schema mappings. Data analysts, who are domain experts but not necessarily have technical data management skills, pose ontology-mediated queries over the conceptualization, which are automatically translated to the appropriate query language for the sources at hand. Following well-established rules when designing schema mappings allows to automate the process of query rewriting and execution.

Overview

Information integration, or data integration, has been an active problem of study for decades. Shortly, it consists in giving a single query involving several data sources to get a single answer.

Semantic Web technologies are well-suited to implement such data integration settings, where given the simplicity and flexibility of ontologies, they constitute an ideal tool to define a unified interface (i.e., *global vocabulary* or *schema*) for such heterogeneous environments. Indeed, the goal of an ontology is precisely to conceptualize the knowledge of a domain; see Gruber (2009). Ontologies are structured into two levels, the TBox and the ABox, where the former represents general properties of concepts and roles (namely, terminology) and the latter represents instances of such concepts and roles (namely, assertions). Such knowledge is commonly represented in terms of the Resource Description Framework (RDF) in the form of triples *subject-predicate-object* (see Wood et al. 2014), which enables to automate its processing and thus opens the door to exchange such information on the Web as Linked Data; see Bizer et al. (2009). Therefore, a vast number of ontologies, or vocabularies, have been proposed to achieve com-



Integration-Oriented Ontology, Fig. 1 Example of query execution in the Semantic Web

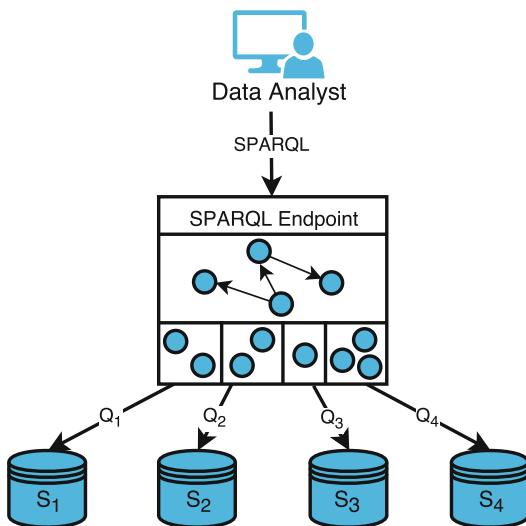
mon consensus when sharing data, such as the RDF Data Cube Vocabulary or the Data Catalog Vocabulary.

Data providers make available their datasets via SPARQL endpoints that implement a protocol where given a SPARQL query, a set of triples is obtained, properly annotated with respect to an ontology. Hence, the data analysts' queries must separately fetch the sets of triples of interest from each endpoint and integrate the results. Furthermore, such settings assume that both TBox and ABox are materialized in the ontology. Figure 1 depicts the Semantic Web architecture, where the data analyst is directly responsible of firstly decomposing the query and issuing the different pieces into several homogeneous SPARQL endpoints and afterward collecting all the results and composing the single required answer.

The alternative architecture for data integration is that of a federated schema that serves as a unique point of entry, which is responsible of mediating the queries to the respective sources. Such federated information systems are commonly formalized as a triple $(\mathcal{G}, \mathcal{S}, \mathcal{M})$, such that \mathcal{G} is the global schema, \mathcal{S} is the source schema, and \mathcal{M} are the schema mappings between \mathcal{G} and \mathcal{S} ; see Lenzerini (2002). With such definition two major problems arise, namely, how to design mappings capable of expressing complex rela-

tionships between \mathcal{G} and \mathcal{S} and how to translate and compose queries posed over \mathcal{G} into queries over \mathcal{S} . The former led to the two fundamental approaches for schema mappings, *global-as-view* (GAV) and *local-as-view* (LAV), while the latter has generically been formulated as the problem of rewriting queries using views; see Halevy (2001) for a survey.

Still, providing an integrated view over a heterogeneous set of data sources is a challenging problem, commonly referred to as the data variety challenge, that traditional data integration techniques fail to address in Big Data contexts given the heterogeneity of formats in data sources; see Horrocks et al. (2016). Current attempts, namely, *ontology-based data access* (OBDA), adopt ontologies to represent \mathcal{G} , as depicted in Fig. 2. In such case, unlike in the Semantic Web, the TBox resides in the ontology as sets of triples, but the ABox in the data sources, in its source format. The most prominent OBDA approaches are based on generic reasoning in description logics (DLs) for query rewriting (see Poggi et al. 2008). They present data analysts with a virtual knowledge graph (i.e., \mathcal{G}), corresponding to the TBox of the domain of interest, in an *OWL2 QL* ontology; see Grau et al. (2012). Such TBox is built upon the *DL-Lite* family of DLs, which allows to represent conceptual models with polynomial



Integration-Oriented Ontology, Fig. 2 Example of query execution in integration-oriented ontologies

cost for reasoning in the TBox; see Calvanese et al. (2007). These rewritings remain tractable as schema mappings follow a GAV approach that characterize concepts in \mathcal{G} in terms of queries over S , providing simplicity in the query answering methods, consisting just of unfolding the queries to express them in terms of the sources. More precisely, given a SPARQL query over \mathcal{G} , it is rewritten by generic reasoning mechanisms into a first-order logic expression and further translated into a union of conjunctive queries over S . Nevertheless, despite the flexibility on querying, the management of the sources is still a problem (magnified in Big Data settings), as the variability in their content and structure would potentially entail reconsidering all existing mappings (a well-known drawback in GAV).

Alternatively to generic reasoning-based approaches, we have vocabulary-based approaches, which are not necessarily limited by the expressiveness of a concrete DL, as they do not rely on generic reasoning algorithms. In such settings, tailored metadata models for specific integration tasks have been proposed, focusing on linking data sources by means of simple annotations with external vocabularies; see Varga et al. (2014). With such simplistic approach, it is possible to define LAV mappings that characterize elements of the source schemata in terms of a query over

the common ontology (facilitating the management of evolution in the sources). However, in the case of LAV, the trade-off comes at the expense of query answering, which now becomes a computationally complex task. Different proposals of specific algorithms exist in the literature for query rewriting under LAV mappings, such as the Bucket algorithm (see Levy et al. 1996) or MiniCon (see Pottinger and Halevy 2001). In the context of integration-oriented ontologies, new specific algorithms must be devised that leverage the information of the metamodel and the annotations to automatically rewrite queries over \mathcal{G} in terms of S .

Key Research Findings

In the line of OBDA, *DL-Lite* set the cornerstone for research, and many extensions have been proposed in order to increase the expressivity of the ontology while maintaining the cost of reasoning for query answering tractable. Some examples of such approaches are the families of description logics *ELH1* (see Pérez-Urbina et al. 2009), *Datalog*[±] (see Gottlob et al. 2011), or *ALC* (see Feier et al. 2017). Attention has also been paid to change management in this context and the definition of a logic able to represent temporal changes in the ontology; see Lutz et al. (2008). Relevant examples include *TQL* that provides a temporal extension of *OWL2 QL* and enables temporal conceptual modeling (see Artale et al. 2013) and a logic that delves on how to provide such temporal aspects for specific attributes (see Keet and Ongoma 2015).

Regarding vocabulary-based approaches purely in the context of data integration, R2RML has been proposed as a standard vocabulary to define mappings from relational databases to RDF (see Cyganiak et al. 2012), and different techniques ad hoc appeared to automatically extract such mappings; see Jiménez-Ruiz et al. (2015).

Examples of Application

Prominent results of generic reasoning OBDA systems are Ontop (see Calvanese et al. 2017),

Grind (see Hansen and Lutz 2017), or Clipper (see Eiter et al. 2012). Ontop provides an end-to-end solution for OBDA, which is integrated with existing Semantic Web tools such as Protégé or Sesame. It assumes a *DL-Lite* ontology for \mathcal{G} , and while processing an ontology-mediated query, it applies several optimization steps. Oppositely, Grind assumes that \mathcal{G} is formulated using the \mathcal{EL} DL, which has more expressivity than *DL-Lite*, but does not guarantee that a first-order logic rewriting exists for the queries. Clipper relies on Horn-SH \mathcal{IQ} ontologies, a DL that extends *DL-Lite* and \mathcal{EL} . All works assume relational sources and translate first-order logic expressions into SQL queries. Nonetheless, recent works present approaches to adopt NOSQL stores as underlying sources, such as MongoDB; see Botoeva et al. (2016).

Regarding vocabulary-based approaches, we can find an RDF metamodel to be instantiated into \mathcal{G} and S for the purpose of accommodating schema evolution in the sources and use RDF named graphs to implement LAV mappings; see Nadal et al. (2018). Such approach simplifies the definition of LAV mappings, for nontechnical users, which has been commonly characterized as a difficult task. Query answering, however, is restricted to only traversals over \mathcal{G} . Another proposal is GEMMS, a metadata management system for Data Lakes, that automatically extracts metadata from the sources to annotate them with respect to the proposed metamodel; see Quix et al. (2016).

Future Directions for Research

Despite that much work has been done in the foundations of OBDA and description logics, there are two factors that complicate the problem in the context of Big Data, namely, size and lack of control over the data sources. Given this lack of control and the heterogeneous nature of data sources, it is needed to further study the kind of mappings and query languages that can be devised for different data models. Nevertheless, this is not enough; new governance mechanisms must be put in place giving rise to Semantic Data Lakes, understood as huge repositories of

disparate files that require advanced techniques to annotate their content and efficiently facilitate querying. Related to this efficiency, more attention must be paid to query optimization and execution, specially when combining independent sources and data streams. Finally, from another point of view, given that the goal of integration-oriented ontologies is to enable access to non-technical users, it is necessary to empower them by providing clear and traceable query plans that, for example, trace provenance and data quality metrics throughout the process.

Cross-References

- ▶ [Data Integration](#)
- ▶ [Federated RDF Query Processing](#)
- ▶ [Ontologies for Big Data](#)
- ▶ [Schema Mapping](#)

References

- Artale A, Kontchakov R, Wolter F, Zakharyashev M (2013) Temporal description logic for ontology-based data access. In: IJCAI 2013, Proceedings of the 23rd international joint conference on artificial intelligence, Beijing, 3–9 Aug 2013, pp 711–717
- Bizer C, Heath T, Berners-Lee T (2009) Linked data – the story so far. Int J Semant Web Inf Syst 5(3):1–22
- Botoeva E, Calvanese D, Cogrel B, Rezk M, Xiao G (2016) OBDA beyond relational Dbs: a study for MongoDB. In: 29th international workshop on description logics
- Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R (2007) Tractable reasoning and efficient query answering in description logics: the *DL-Lite* family. J Autom Reason 39(3):385–429
- Calvanese D, Cogrel B, Komla-Ebri S, Kontchakov R, Lanti D, Rezk M, Rodriguez-Muro M, Xiao G (2017) Ontop: answering SPARQL queries over relational databases. Semantic Web 8(3):471–487
- Cyganiak R, Das S, Sundara S (2012) R2RML: RDB to RDF mapping language. W3C recommendation, W3C. <http://www.w3.org/TR/2012/REC-r2rml-20120927/>
- Eiter T, Ortiz M, Simkus M, Tran T, Xiao G (2012) Query rewriting for Horn-SHIQ plus rules. In: Proceedings of the twenty-sixth AAAI conference on artificial intelligence, Toronto, 22–26 July 2012
- Feier C, Kuusisto A, Lutz C (2017) Rewritability in monadic disjunctive datalog, MMSNP, and expressive description logics (invited talk). In: 20th international conference on database theory, ICDT 2017, Venice, 21–24 Mar 2017, pp 1:1–1:17

- Gottlob G, Orsi G, Pieris A (2011) Ontological queries: rewriting and optimization. In: Proceedings of the 27th international conference on data engineering, ICDE 2011, 11–16 Apr 2011, Hannover, pp 2–13
- Grau BC, Fokoue A, Motik B, Wu Z, Horrocks I (2012) OWL 2 web ontology language profiles, 2nd edn. W3C recommendation, W3C. <http://www.w3.org/TR/2012/REC-owl2-profiles-20121211>
- Gruber T (2009) Ontology. In: Tamer Özsu M, Liu L (eds) Encyclopedia of database systems. Springer, New York/London, pp 1963–1965
- Halevy AY (2001) Answering queries using views: a survey. VLDB J 10(4):270–294
- Hansen P, Lutz C (2017) Computing FO-rewritings in EL in practice: from atomic to conjunctive queries. In: 16th international semantic web conference (ISWC), pp 347–363
- Horrocks I, Giese M, Kharlamov E, Waaler A (2016) Using semantic technology to tame the data variety challenge. IEEE Internet Comput 20(6):62–66
- Jiménez-Ruiz E, Kharlamov E, Zheleznyakov D, Horrocks I, Pinkel C, Skjæveland MG, Thorstensen E, Mora J (2015) Bootox: practical mapping of RDBs to OWL 2. In: The semantic web – ISWC 2015 – 14th international semantic web conference, Bethlehem, 11–15 Oct 2015, Proceedings, Part II, pp 113–132
- Keet CM, Ongoma EAN (2015) Temporal attributes: status and subsumption. In: 11th Asia-Pacific conference on conceptual modelling, APCCM 2015, Sydney, pp 61–70
- Lenzerini M (2002) Data integration: a theoretical perspective. In: 21st ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS), pp 233–246
- Levy AY, Rajaraman A, Ordille JJ (1996) Querying heterogeneous information sources using source descriptions. In: 22th international conference on very large data bases (VLDB), pp 251–262
- Lutz C, Wolter F, Zakharyashev M (2008) Temporal description logics: a survey. In: 15th international symposium on temporal representation and reasoning, TIME 2008, Université du Québec à Montréal, 16–18 June 2008, pp 3–14
- Nadal S, Romero O, Abelló A, Vassiliadis P, Vansumermen S (2018) An integration-oriented ontology to govern evolution in big data ecosystems. Information Systems, Elsevier. <https://doi.org/10.1016/j.is.2018.01.006>
- Pérez-Urbina H, Motik B, Horrocks I (2009) A comparison of query rewriting techniques for dl-lite. In: Proceedings of the 22nd international workshop on description logics (DL 2009), Oxford, 27–30 July 2009
- Poggi A, Lembo D, Calvanese D, De Giacomo G, Lenzerini M, Rosati R (2008) Linking data to ontologies. J Data Semant 10:133–173
- Pottinger R, Halevy AY (2001) Minicon: a scalable algorithm for answering queries using views. VLDB J 10(2–3):182–198
- Quix C, Hai R, Vatov I (2016) GEMMS: a generic and extensible metadata management system for data lakes. In: 28th international conference on advanced information systems engineering (CAiSE), pp 129–136, CAiSE Forum
- Varga J, Romero O, Pedersen TB, Thomsen C (2014) Towards next generation BI systems: the analytical metadata challenge. In: 16th international conference on data warehousing and knowledge discovery (DaWaK), pp 89–101
- Wood D, Cyganiak R, Lanthaler M (2014) RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C. <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>
-
- ## Integrity
- [Security and Privacy in Big Data Environment](#)
-
- ## Intelligent Disk
- [Active Storage](#)
-
- ## Interactive Visualization
- [Big Data Visualization Tools](#)
-
- ## Introduction to Stream Processing Algorithms
- Nicoló Rivetti
Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel
-
- ## Definitions
- Data streaming focuses on estimating functions over streams, which is an important task in data-intensive applications. It aims at approximating functions or statistical measures over (distributed) massive stream(s) in polylogarithmic space over the size and/or the domain

size (i.e., number of distinct items) of the stream(s).

Overview

Many different domains are concerned by the analysis of streams, including machine learning, data mining, databases, information retrieval, and network monitoring. In all these fields, it is necessary to quickly and precisely process a huge amount of data. This can also be applied to any other data issued from distributed applications such as social networks or sensor networks. Given these settings, the real-time analysis of large streams, relying on full-space algorithms, is often not feasible. Two main approaches exist to monitor massive data streams in real time with small amount of resources: sampling and summaries.

Computing information theoretic measures in the data stream model is challenging essentially because one needs to process a huge amount of data sequentially, on the fly, and by using very little storage with respect to the size of the stream. In addition, the analysis must be robust over time to detect any sudden change in the observed streams (which may be an indicator of some malicious behavior).

Data Streaming Models

Let (Muthukrishnan 2005) consider a stream $\sigma = \langle t_1, \dots, t_j, \dots, t_m \rangle$, where each item t is drawn from a universe of size n . More in details, t denotes the value of the item in the set $[n] = \{1, \dots, n\}$, while the subscript j denotes the position in the stream in the index sequence $[m] = \{1, \dots, m\}$. Then m is the size (or number of items) of the stream σ , and n is the number of distinct items. Notice that in general, both n and m are large (e.g., network traffic) and unknown.

The stream σ implicitly defines a frequency vector $\mathbf{f} = \langle f_1, \dots, f_t, \dots, f_n \rangle$ where f_t is the frequency of item t in the stream σ , i.e., the number of occurrences of item t in σ . The empirical probability distribution is defined as $\mathbf{p} = \langle p_1, \dots, p_t, \dots, p_n \rangle$, where $p_t = f_t/m$ is the empirical probability of occurrence of item

t in the stream σ . Our aim is to compute some function ϕ on σ . In this model we are allowed only to access the sequence in its given order (no random access), and the function must be computed in a single pass (online). The challenge is to achieve this computation in sublinear space and time with respect to m and n . In other words, the space and time complexities should be at most $smallO(\min\{m, n\})$, but the goal is $O(\log(m \cdot n))$. However, since reaching the latter bound is quite hard (if not impossible), one deals in general with complexities such as $polylog(m, n)$. In general, $\phi(x)$ is said to be $polylog(x)$ if $\exists C > 0$, $\phi(x) = O(\log^C x)$. By abusing the notation, we then define $polylog(m, n)$ as $\exists C, C' > 0$ such that $\phi(m, n) = O(\log^C m + \log^{C'} n)$.

Sliding window model — In many applications, there is a greater interest in considering only the “recently observed” items than the whole semi-infinite stream. This gives rise to the *sliding window* model (Datar et al. 2002): items arrive continuously and expire after exactly M steps. The relevant data is then the set of items that arrived in the last M steps, i.e., the *active window*. A step can either be defined as a sample arrival, then the active window would contain exactly M items, and the window is said to be count-based. Otherwise, the step is defined as a time tick, either logical or physical, and the window is said to be time-based. More formally, let us consider the count-based sliding window and recall that by definition, t_j is the item arriving at step j . In other words, the stream σ at step m is the sequence $\langle t_1, \dots, t_{m-M}, t_{m-M+1}, \dots, t_m \rangle$. Then, the desired function ϕ has to be evaluated over the stream $\sigma(M) = \langle t_{m-M+1}, \dots, t_m \rangle$. Notice that if $M = m$, the sliding window model boils down to the classical data streaming model. Two related models may be named as well: the landmark window and jumping window models. In the former, a step (landmark) is chosen, and the function is evaluated on the stream following the landmark. In the jumping window model, the sliding window is split into sub-windows and moves forward only when the most recent sub-window is full (i.e., it jumps forward). In general,

solving a problem in the sliding window model is harder than in the classical data streaming model: using little memory (low space complexity) implies some kind of data aggregation. The problem is then how to slide the window forward, i.e., how to remove the updates performed by the items that exited the window, and how to introduce new items. For instance, *basic counting*, i.e., count the number of 1 in the stream of bits, is a simple problem whose solution is a prerequisite for efficient maintenance of a variety of more complex statistical aggregates. While this problem is trivial for the data streaming model, it becomes a challenge in the sliding windowed model (Datar et al. 2002).

Distributed streaming model and functional monitoring — In large distributed systems, it is most likely critical to gather various aggregates over data spread across a large number of nodes. This can be modeled by a set of nodes, each observing a stream of items. These nodes collaborate to evaluate a given function over the global distributed stream. For instance, current network management tools analyze the input streams of a set of routers to detect malicious sources or to extract user behaviors (Anceaume and Busnel 2014; Ganguly et al. 2007).

The *distributed streaming* model (Gibbons and Tirthapura 2001) considers a set of nodes making observations and cooperating, through a *coordinator*, to compute a function over the union of their observations. More formally, let \mathcal{I} be a set of s nodes $S_1, \dots, S_i, \dots, S_s$ that do not communicate directly between them but only with a coordinator (i.e., a central node or base station). The communication channel can be either one way or bidirectional. The *distributed* stream σ is the union the sub-streams $\sigma_1, \dots, \sigma_i, \dots, \sigma_s$ locally read by the s nodes, i.e., $\sigma = \bigcup_{i=1}^s \sigma_i$, where $\sigma_i \cup \sigma_j$ is a stream containing all the items of σ_i and σ_j with any interleaving. The generic stream σ_i is the input to the generic node S_i . Then we have $m = \sum_{i=1}^s m_i$, where m_i is the size (or number of items) of the i -th sub-stream σ_i of the distributed stream σ . Each sub-stream σ_i implicitly defines a frequency vector

$\mathbf{f}_i = \langle f_{1,i}, \dots, f_{t,i}, \dots, f_{n,i} \rangle$ where $f_{t,i}$ is the frequency of item t in the sub-stream σ_i , i.e., the number of occurrences of item t in the sub-stream σ_i . Then, we also have $f_t = \sum_{i=1}^s f_{t,i}$. The empirical probability distribution is defined as $\mathbf{p}_i = \langle p_{1,i}, \dots, p_{t,i}, \dots, p_{n,i} \rangle$, where $p_{t,i} = f_{t,i}/m_i$ is the empirical probability of occurrence of item t in the stream σ_i .

With respect to the classical data streaming model, we want to compute the function $\phi(\sigma) = \phi(\sigma_1 \cup \dots \cup \sigma_s)$. This model introduces the additional challenge to minimize the communication complexity (in bits), i.e., the amount of data exchanged between the s nodes and the coordinator.

The *distributed functional monitoring* problem (Cormode et al. 2011; Cormode 2011) has risen only in the 2000s. With respect to the distributed streaming model, functional monitoring requires a continuous evaluation of $\phi(\sigma) = \phi(\sigma_1 \cup \dots \cup \sigma_s)$ at each time instant $j \in [m]$. Bounding ϕ to be monotonic, it is possible to distinguish two cases: *threshold* monitoring (i.e., raising an alert if $\phi(\sigma)$ exceed a threshold *heta*) and *value* monitoring (i.e., approximating $\phi(\sigma)$). It then can be shown (Cormode et al. 2011) that value monitoring can be reduced with a factor $O(1/\varepsilon \log R)$ (where R is the largest value reached by the monitored function) to threshold monitoring. However, this reduction does not hold any more in a more general setting with non-monotonic function (e.g., entropy). Notice that to minimize the communication complexity and maximize the accuracy, the algorithms designed in this model tend to minimize the communication when nothing relevant happens while reacting as fast as possible in response to outstanding occurrences or patterns.

Building Blocks

Approximation and randomization — To compute ϕ on σ in these models, most of the research relies on ε or (ε, δ) approximations. An algorithm A that evaluates the stream σ in a single pass (online) is said to be an (ε, δ) additive approximation of the function ϕ on a stream σ if, for any prefix of size m of the input stream σ , the

output $\hat{\phi}$ of A is such that $\mathbb{P}\{|\hat{\phi} - \phi| > \varepsilon\psi\} \leq \delta$, where $\varepsilon, \delta > 0$ are given as precision parameters and ψ is an arbitrary function (its parameter may be n and/or m). Similarly, an algorithm A that evaluates the stream σ in a single pass (online) is said to be an (ε, δ) approximation of the function ϕ on a stream σ if, for any prefix of size m of the input stream σ , the output $\hat{\phi}$ of A is such that $\mathbb{P}\{|\hat{\phi} - \phi| > \varepsilon\phi\} \leq \delta$, where $\varepsilon, \delta > 0$ are given as precision parameters. Finally, if $\delta = 0$, independently of the algorithm parametrization, then A is deterministic, and we drop the δ from the approximation notation.

Sampling — The sampling approach reads once all the items; however, only a few are kept for the computation. It is also current that some metadata are associated with the retained items. The aim is for a poly-logarithmic storage size with respect to size of the input stream. There are several methods, both deterministic and randomized, to choose which subset of items has to be kept (Muthukrishnan 2005; Alon et al. 1996; Misra and Gries 1982). This allows to exactly compute functions on these samples, which are expected to be representative. However, the accuracy of this computation, with respect to the stream in its entirety, fully depends on the volume of data that has been sampled and the location of the sampled data. Worse, an adversary may easily take advantage of the sampling policy to arbitrarily bias the sample itself.

Summaries, random projections, and sketches

— The summary approach consists in scanning on the fly each piece of data of the input stream and keep locally only compact synopses or sketches containing the most important information about data items. This approach enables to derive some data streams statistics with guaranteed error bounds. Random projections produce a storage size reduction, leveraging pseudorandom vectors and projections. The pseudorandom vector can be generated using space-efficient computation over limitedly independent random variables (Charikar et al.

2002; Cormode and Muthukrishnan 2005). A relevant class of projections are sketches. A data structure DS is a sketch if there is a space-efficient combining algorithm COMB such that, for every two streams σ_1 and σ_2 , we have $COMB(DS(\sigma_1), DS(\sigma_2)) = DS(\sigma_1 \cup \sigma_2)$, where $\sigma_1 \cup \sigma_2$ is a stream containing all the items of σ_1 and σ_2 in any order. In order to extend the sketch definition to the sliding window model, we limit the reordering of the items in σ_1 and σ_2 to the last M items (i.e., the active window).

Key Research Findings

There are several relevant results in the data streaming literature, including the computation of the number of distinct items (Bar-Yossef et al. 2002; Flajolet and Martin 1985; Kane et al. 2010), the frequency moments (Alon et al. 1996; Cormode et al. 2011), the most frequent data items (Cormode and Muthukrishnan 2005; Metwally et al. 2005; Misra and Gries 1982), similarity distance between two streams (Anceaume and Busnel 2014), and the entropy of the stream (Chakrabarti et al. 2007). For the sake of brevity, this section is limited to the classical problem of frequency estimation.

Estimating the frequency moments is one of the first problems to be dealt with in a streaming fashion, namely, in the seminal paper (Alon et al. 1996). The i -th frequency moment is defined as $F_i = \sum_{j=1}^n f_j^i = \|f\|_i$. F_0 is then the number of distinct items of the stream (assuming that $0^0 = 0$), and F_1 is the length m of the stream. The frequency moments for F_i where $i \geq 2$ indicate the skew of the data, which is a relevant information in many distributed applications.

The *frequency estimation* problem is closely related; however it does not provide a single aggregate value to represent the whole stream but a more detailed view. In particular, given a stream $\sigma = \langle t_1, \dots, t_m \rangle$, the frequency estimation problem is to provide an estimate \hat{f}_t of f_t for each item $t \in [n]$. In other words, the task is to build an approximation of the frequency distribution of the stream. Estimating the frequency of any piece of information in large-scale distributed

data streams became of utmost importance in the last decade (e.g., in the context of network monitoring, Big Data, *etc*). IP network management, among others, is a paramount field in which the ability to estimate the stream frequency distribution in a single pass (i.e., online) and quickly can bring huge benefits. For instance, it could rapidly detect the presence of anomalies or intrusions identifying sudden changes in the communication patterns.

Data streaming model — Considering the data streaming model, the Count-Sketch algorithm (Charikar et al. 2002) was an initial result satisfying the following accuracy bound: $\mathbb{P}[\|\hat{f}_t - f_t\| \geq \varepsilon \|f_{-t}\|_2] \leq \delta$, where f_{-t} represents the frequency vector f without the scalar f_t (i.e., $|f_{-t}| = n - 1$).

Few years later, a similar algorithm, Count-Min sketch (Cormode and Muthukrishnan 2005), has been presented. The bound on the estimation accuracy of the Count-Min is weaker than the one provided by the Count-Sketch, in particular, for skewed distribution. On the other hand, the Count-Min shaves a $1/\varepsilon$ factor from the space complexity and halves the hash functions. The Count-Min consists of a two-dimensional matrix \mathfrak{F} of size $r \times c$, where $r = \lceil \log 1/\delta \rceil$ and $c = \lceil e/\varepsilon \rceil$, where e is Euler's constant. Each row is associated with a different two-universal hash function $h_{i \in [r]} : [n] \rightarrow [c]$. For each item t_j , it updates each row: $\forall i \in [r], \mathfrak{F}[i, h_i(t)] \leftarrow \mathfrak{F}[i, h_i(t)] + 1$. That is, the entry value is the sum of the frequencies of all the items mapped to that entry. Since each row has a different collision pattern, upon request of \hat{f}_t , we want to return the entry associated with t minimizing the collisions impact. In other words, the algorithm returns, as f_t estimation, the entry associated with t with the lowest value: $\hat{f}_t = \min_{i \in [r]} \{\mathfrak{F}[i, h_i(t)]\}$. Listing 1 presents the global behavior of the Count-Min algorithm. The space complexity of this algorithm is $O(\frac{1}{\varepsilon} \log \frac{1}{\delta} (\log m + \log n))$ bits, while the update time complexity is $O(\log 1/\delta)$. Therefore, the following quality bounds hold: $\mathbb{P}[\|\hat{f}_t - f_t\| \geq \varepsilon \|f_{-t}\|_1] \leq \delta$, while $f_t \leq \hat{f}_t$ is always true.

Sliding window model — There are two extensions of the Count-Min algorithm to the sliding window model. The ECM-Sketch replaces the entries of the Count-Min matrix with instances of the wave data structure (Gibbons and Tirthapura 2014) which provides an ($\varepsilon_{\text{wave}}$) approximation for the basic counting problem in the sliding window model. A more recent solution (Rivetti et al. 2015a) superseded the ECM-Sketch performances. It has a similar approach, replacing the entries with a novel data structure to keep track of the changes in the frequency distribution of the stream over time.

Applications

Distributed Denial of Service Detection

Network monitoring, as mentioned previously, is one of the classical fields of application given the natural fitting of the data streaming models to network traffics. The detection of distributed denial of service (DDoS) can be reduced to the *distributed heavy hitters* problem, having first been studied in Manjhi et al. (2005), by making the assumption that the attack is locally detectable by routers. Further solutions relaxed the model but relied on other assumptions such as the existence of a known gap between legit and attack traffic (Zhao et al. 2010), a priori knowledge of the IP addresses distribution (Zhao et al. 2006) (i.e., distributed bag model). Finally, Yi and Zhang (2013) propose a solution in the spirit of the distributed functional monitoring model to minimize the communication cost between the nodes and the coordinator. However each router maintains at any time the exact frequency of each IP address in its local network stream, i.e., their solution is extremely memory expensive. A recent solution (Anceaume et al. 2015), based on the Count-Min, detects distributed denial of service without relying on any assumption on the stream characteristics.

Stream Processing Optimization

Data streaming has also been extensively applied to databases, for instance, providing efficient estimation of the join size (Alon et al. 1996;

Listing 1: Count-Min Sketch Algorithm.

```

1: init ( $r, c$ ) do
2:    $\mathfrak{F}[1 \dots r, 1 \dots c] \leftarrow 0_{r,c}$ 
3:    $h_1, \dots, h_r : [n] \rightarrow [c]$ 
4: end init
5: function UPDATE( $tj$ )
6:   for  $i = 1$  to  $r$  do
7:      $\mathfrak{F}[i, h_i(t)] \leftarrow \mathfrak{F}[i, h_i(t)] + 1$ 
8: function GETFREQ( $t$ )
9:   return  $\min\{\mathfrak{F}[i, h_i(t)] \mid 1 \leq i \leq r\}$ 

```

▷ r hash functions from a 2-universal family.
 ▷ reads item tj from the stream σ
 ▷ returns \widehat{f}_t

Vengerov et al. 2015). There are many query plan optimizations performed in database systems that can leverage such estimations.

A more recent application field is stream processing (Hirzel et al. 2014) and complex event processing. Load balancing in distributed computing is a well-known problem that has been extensively studied since the 1980s (Cardellini et al. 2002). Two ways to perform load balancing can be identified in stream processing systems (Hirzel et al. 2014): either when placing the operators on the available machines (Carney et al. 2003; Cardellini et al. 2016) or when assigning load to the parallelized instances of operator O . If O is stateless, then tuples are randomly assigned to instances (*random* or *shuffle grouping*). Extending the Count-Min (Cormode and Muthukrishnan 2005), Rivetti et al. (2016a) presents an algorithm aimed at reducing the overall tuple completion time in shuffle-grouped stages. Key grouping is another grouping technique used by stream processing frameworks to simplify the development of parallel stateful operators. Through key grouping a stream of tuples is partitioned in several disjoint sub-streams depending on the values contained in the tuples themselves. Each operator instance target of one sub-stream is guaranteed to receive all the tuples containing a specific key value. A common solution to implement key grouping is through hash functions that, however, are known to cause load imbalances on the target operator instances when the input data stream is characterized by a skewed value distribution. Gedik (2014) proposes a solution to this problem leveraging consistent hashing and the lossy counting (Manku and Motwani 2002) algorithm. In Rivetti et al. (2015b) the authors propose an improvement over (Gedik 2014) using

the space-saving (Metwally et al. 2005) algorithm instead. The basic intuition is that heavy hitters induce most of the unbalance in the load distribution. Then, in both works the authors leverage solutions to the heavy hitter problem to isolate/carefully handle the keys that have a frequency larger than/close to $1/k$. On the other hand, Nasir et al. (2015) applied the power of two choices approach to this problem, i.e., relax the key grouping constraints by spreading each key on two instances, thus restricting the applicability of their solution but also achieving better balancing.

Caneill et al. (2016) present another work focusing on load balancing through a different angle. In topologies with sequences of stateful operators, and thus key-grouped stages, the likelihood that a tuple is sent to different operator instances located on different machines is high. Since network is a bottleneck for many stream applications, this behavior significantly degrades their performance. The authors use the space-saving (Metwally et al. 2005) algorithm to identify the heavy hitters and analyze the correlation between keys of subsequent key-grouped stages. This information is then leveraged to build a key-to-instance mapping that improves stream locality for stateful stream processing applications.

The *load-shedding* (or *admission control*) technique (Hirzel et al. 2014) goal is to drop some of the input in order to keep the input load under the maximum capacity of the system. Without load shedding, the load would pile up inducing buffer overflows and trashing. In Rivetti et al. (2016b) the authors provide a novel load-shedding algorithm based on an extension of the Count-Min (Cormode and Muthukrishnan 2005), targeted at stream processing systems.

References

- Alon N, Matias Y, Szegedy M (1996) The space complexity of approximating the frequency moments. In: Proceedings of the 28th ACM symposium on theory of computing, STOC
- Anceaume E, Busnel Y (2014) A distributed information divergence estimation over data streams. *IEEE Trans Parallel Distrib Syst* 25(2):478–487
- Anceaume E, Busnel Y, Rivetti N, Sericola B (2015) Identifying global icebergs in distributed streams. In: Proceedings of the 34th international symposium on reliable distributed systems, SRDS
- Bar-Yossef Z, Jayram TS, Kumar R, Sivakumar D, Trevisan L (2002) Counting distinct elements in a data stream. In: Proceedings of the 6th international workshop on randomization and approximation techniques, RANDOM
- Caneill M, El Rhedane A, Leroy V, De Palma N (2016) Locality-aware routing in stateful streaming applications. In: Proceedings of the 17th international middleware conference, Middleware’16
- Cardellini V, Casalicchio E, Colajanni M, Yu PS (2002) The state of the art in locally distributed web-server systems. *ACM Comput Surv* 34(2):263–311
- Cardellini V, Grassi V, Lo Presti F, Nardelli M (2016) Optimal operator placement for distributed stream processing applications. In: Proceedings of the 10th ACM international conference on distributed and event-based systems, DEBS
- Carney D, Çetintemel U, Rasin A, Zdonik S, Cherniack M, Stonebraker M (2003) Operator scheduling in a data stream manager. In: Proceedings of the 29th international conference on very large data bases, VLDB
- Chakrabarti A, Cormode G, McGregor A (2007) A near-optimal algorithm for computing the entropy of a stream. In: Proceedings of the 18th ACM-SIAM symposium on discrete algorithms, SODA
- Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Proceedings of the 29th international colloquium on automata, languages and programming, ICALP
- Cormode G (2011) Continuous distributed monitoring: a short survey. In: Proceedings of the 1st international workshop on algorithms and models for distributed event processing, AlMoDEP’11
- Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. *J Algorithms* 55(1):58–75
- Cormode G, Muthukrishnan S, Yi K (2011) Algorithms for distributed functional monitoring. *ACM Trans Algorithms* 7(2):21:1–21:20
- Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. *SIAM J Comput* 31(6):1794–1813
- Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. *J Comput Syst Sci* 31(2):182–209
- Ganguly S, Garafalakis M, Rastogi R, Sabnani K (2007) Streaming algorithms for robust, real-time detection of DDoS attacks. In: Proceedings of the 27th international conference on distributed computing systems, ICDCS
- Gedik B (2014) Partitioning functions for stateful data parallelism in stream processing. *The VLDB J* 23(4): 517–539
- Gibbons PB, Tirthapura S (2001) Estimating simple functions on the union of data streams. In: Proceedings of the 13th ACM symposium on parallel algorithms and architectures, SPAA
- Gibbons PB, Tirthapura S (2004) Distributed streams algorithms for sliding windows. *Theory Comput Syst* 37(3):457–478
- Hirzel M, Soulé R, Schneider S, Gedik B, Grimm R (2014) A catalog of stream processing optimizations. *ACM Comput Surv* 46(4):41–34
- Kane DM, Nelson J, Woodruff DP (2010) An optimal algorithm for the distinct elements problem. In: Proceedings of the 19th ACM symposium on principles of database systems, PODS
- Manjhi A, Shkapenyuk V, Dhamdhere K, Olston C (2005) Finding (recently) frequent items in distributed data streams. In: Proceedings of the 21st international conference on data engineering, ICDE
- Manku G, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of the 28th international conference on very large data bases, VLDB
- Metwally A, Agrawal D, El Abbadi A (2005) Efficient computation of frequent and top-k elements in data streams. In: Proceedings of the 10th international conference on database theory, ICDT
- Misra J, Gries D (1982) Finding repeated elements. *Sci Comput Program* 2:143–152
- Muthukrishnan S (2005) Data streams: algorithms and applications. Now Publishers Inc., Hanover
- Nasir MAU, Morales GDF, Soriano DG, Kourtellis N, Serafini M (2015) The power of both choices: practical load balancing for distributed stream processing engines. In: Proceedings of the 31st IEEE international conference on data engineering, ICDE
- Rivetti N, Busnel Y, Mostefaoui A (2015a) Efficiently summarizing distributed data streams over sliding windows. In: Proceedings of the 14th IEEE international symposium on network computing and applications, NCA
- Rivetti N, Querzoni L, Anceaume E, Busnel Y, Sericola B (2015b) Efficient key grouping for near-optimal load balancing in stream processing systems. In: Proceedings of the 9th ACM international conference on distributed event-Based systems, DEBS
- Rivetti N, Anceaume E, Busnel Y, Querzoni L, Sericola B (2016a) Proactive online scheduling for shuffle grouping in distributed stream processing systems. In: Proceedings of the 17th ACM/IFIP/USENIX international middleware conference, Middleware
- Rivetti N, Busnel Y, Querzoni L (2016b) Load-aware shedding in stream processing systems. In: Proceedings of the 10th ACM international conference on distributed event-based systems, DEBS

- Vengerov D, Menck AC, Zait M, Chakkappen SP (2015) Join size estimation subject to filter conditions. *Proc VLDB Endow* 8(12):1530–1541
- Yi K, Zhang Q (2013) Optimal tracking of distributed heavy hitters and quantiles. *Algorithmica* 65:206–223
- Zhao Q, Lall A, Ogihsara M, Xu J (2010) Global iceberg detection over distributed streams. In: Proceedings of the 26th IEEE international conference on data engineering, ICDE
- Zhao Q, Ogihsara M, Wang H, Xu J (2006) Finding global icebergs over distributed data sets. In: Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, PODS

Inverted Index Compression

Giulio Ermanno Pibiri and Rossano Venturini
Department of Computer Science, University of Pisa, Pisa, Italy

Definitions

The data structure at the core of nowadays large-scale search engines, social networks, and storage architectures is the *inverted index*. Given a collection of documents, consider for each distinct term t appearing in the collection the integer sequence ℓ_t , listing in sorted order all the identifiers of the documents (docIDs in the following) in which the term appears. The sequence ℓ_t is called the *inverted list* or *posting list* of the term t . The inverted index is the collection of all such lists.

The scope of the entry is the one of surveying the most important encoding algorithms developed for efficient inverted index compression and fast retrieval.

Overview

The inverted index owes its popularity to the efficient resolution of *queries*, expressed as a set of terms $\{t_1, \dots, t_k\}$ combined with a query operator. The simplest operators are Boolean AND and OR. For example, given an AND query, the index has to report *all* the docIDs of the documents containing the terms $\{t_1, \dots, t_k\}$. This operation ultimately boils down to *intersecting*

the inverted lists corresponding to the terms of the query set. Efficient list intersection relies on the operation $\text{NextGEQ}_t(x)$, which returns the integer $z \in \ell_t$ such that $z \geq x$. This primitive is used because it permits to *skip* over the lists to be intersected.

Because of the many documents indexed by search engines and stringent performance requirements dictated by the heavy load of user queries, the inverted lists often store several millions (even billion) of integers and must be searched efficiently. In this scenario, *compressing* the inverted lists of the index appears as a *mandatory* design phase since it can introduce a twofold advantage over a non-compressed representation: feed faster memory levels with more data in order to speed up the query processing algorithms and reduce the number of storage machines needed to host the whole index. This has the potential of letting a query algorithm work in internal memory, which is faster than the external memory system by several orders of magnitude.

Chapter Notation

All logarithms are binary, i.e., $\log x = \log_2 x$, $x > 0$. Let $B(x)$ represent the binary representation of the integer x and $U(x)$ its *unary* representation, that is, a run of x zeroes plus a final one: $0^x 1$. Given a binary string B , let $|B|$ represent its length in bits. Given two binary strings B_1 and B_2 , let $B_1 B_2$ be their concatenation. We indicate with $S(n, u)$ a sequence of n integers drawn from a universe of size u and with $S[i, j]$ the subsequence starting at position i and including endpoint $S[j]$.

Key Research Findings

Integer Compressors

The compressors we consider in this subsection are used to represent a single integer. The most classical solution is to assign each integer a *self-delimiting* (or *uniquely decodable*) variable-length code so that the whole integer sequence is the result of the juxtaposition of the codes of all its integers. Clearly, the aim of such encoders is to

assign the smallest code word as possible in order to minimize the number of bits used to represent the sequence.

In particular, since we are dealing with inverted lists that are monotonically increasing by construction, we can subtract from each element the previous one (the first integer is left as it is), making the sequence be formed by integers greater than zero known as *delta*-gaps (or just *d*-gaps). This popular *delta-encoding* strategy helps in reducing the number of bits for the codes. Most of the literature assumes this sequence form, and, as a result, compressing such sequences of *d*-gaps is a fundamental problem that has been studied for decades.

Elias' Gamma and Delta

The two codes we now describe have been introduced by Elias (1975) in the 1960s. Given an integer $x > 0$, $\gamma(x) = 0^{|B(x)|-1}B(x)$, where $|B(x)| = \lceil \log(x+1) \rceil$. Therefore, $|\gamma(x)| = 2\lceil \log(x+1) \rceil - 1$ bits. Decoding $\gamma(x)$ is simple: first count the number of zeroes up to the one, say there are n of these, then read the following $n+1$ bits, and interpret them as x .

The key inefficiency of γ lies in the use of the unary code for the representation of $|B(x)| - 1$, which may become very large for big integers. The δ code replaces the unary part $0^{|B(x)|-1}$ with $\gamma(|B(x)|)$, i.e., $\delta(x) = \gamma(|B(x)|)B(x)$. Notice that, since we are representing with γ the quantity $|B(x)| = \lceil \log(x+1) \rceil$ which is guaranteed to be greater than zero, δ can represent value zero too. The number of bits required by $\delta(x)$ is $|\gamma(|B(x)|)| + |B(x)|$, which is $2\lceil \log(\lceil |B(x)| + 1 \rceil) \rceil + \lceil \log(x+1) \rceil - 1$.

The decoding of δ codes follows automatically from the one of γ codes.

Golomb-Rice

Rice and Plaunt (1971) introduced a parameter code that can better compress the integers in a sequence if these are highly concentrated around some value. This code is actually a special case of the Golomb code (Golomb 1966), hence its name.

The Rice code of x with parameter k , $R_k(x)$, consists in two pieces: the quotient $q = \lfloor \frac{x-1}{2^k} \rfloor$

and the remainder $r = x - q \times 2^k - 1$. The quotient is encoded in unary, i.e., with $U(q)$, whereas the remainder is written in binary with k bits. Therefore, $|R_k(x)| = q+k+1$ bits. Clearly, the closer 2^k is to the value of x the smaller the value of q : this implies a better compression and faster decoding speed.

Given the parameter k and the constant 2^k that is computed ahead, decoding Rice codes is simple too: count the number of zeroes up to the one, say there are q of these, then multiply 2^k by q , and finally add the remainder, by reading the next k bits. Finally, add one to the computed result.

Variable-Byte

The codes described so far are also called *bit-aligned* as they do not represent an integer using a multiple of a fixed number of bits, e.g., a byte. The decoding speed can be slowed down by the many operations needed to decode a single integer. This is a reason for preferring *byte-aligned* or even word-aligned codes when decoding speed is the main concern.

Variable-Byte (VByte) (Salomon 2007) is the most popular and simplest byte-aligned code: the binary representation of a nonnegative integer is split into groups of 7 bits which are represented as a sequence of bytes. In particular, the 7 least significant bits of each byte are reserved for the data, whereas the most significant (the 8-th), called the *continuation bit*, is equal to 1 to signal continuation of the byte sequence. The last byte of the sequence has its 8-th bit set to 0 to signal, instead, the termination of the byte sequence. The main advantage of VByte codes is decoding speed: we just need to read one byte at a time until we find a value smaller than 128. Conversely, the number of bits to encode an integer cannot be less than 8; thus VByte is only suitable for large numbers, and its compression ratio may not be competitive with the one of bit-aligned codes for small integers.

Various enhancements have been proposed in the literature to improve the (sequential) decoding speed of VByte. This line of research focuses on finding a suitable format of control and data streams in order to reduce the prob-

ability of a branch misprediction that leads to higher instruction throughput and helps keeping the CPU pipeline fed with useful instructions (Dean 2009) and on exploiting the parallelism of SIMD instructions (single-instruction-multiple-data) of modern processors (Stepanov et al. 2011; Plaisance et al. 2015; Lemire et al. 2018).

List Compressors

Differently from the compressors introduced in the previous subsection, the compressors we now consider encode a whole integer list, instead of representing each single integer separately. Such compressors often outperform the ones described before for sufficiently long inverted lists because they take advantage of the fact that inverted lists often contain *clusters* of close docIDs, e.g., runs of consecutive integers, that are far more compressible with respect to highly scattered sequences. The reason for the presence of such clusters is that the indexed documents themselves tend to be clustered, i.e., there are subsets of documents sharing the very same set of terms. Therefore, not surprisingly, list compressors greatly benefit from reordering strategies that focus on reassigning the docIDs in order to form larger clusters of docIDs.

An amazingly simple strategy, but very effective for Web pages, is to assign identifiers to documents according to the lexicographical order of their URLs (Silvestri 2007). A recent approach has instead adopted a recursive graph bisection algorithm to find a suitable reordering of docIDs (Dhulipala et al. 2016). In this model, the input graph is a bipartite graph in which one set of vertices represents the terms of the index and the other set represents the docIDs. Since a graph bisection identifies a permutation of the docIDs, the goal is the one of finding, at each step of recursion, the bisection of the graph which minimizes the size of the graph compressed using delta-encoding.

Block-Based

A relatively simple approach to improve both compression ratio and decoding speed is to encode a *block* of contiguous integers. This line of

work finds its origin in the so-called frame of reference (FOR) (Goldstein et al. 1998).

Once the sequence has been partitioned into blocks (of fixed or variable length), then each block is encoded separately. An example of this approach is *binary packing* (Anh and Moffat 2010; Lemire and Boytsov 2013), where blocks of fixed length are used, e.g., 128 integers. Given a block $S[i, j]$, we can simply represent its integers using a universe of size $\lceil \log(S[j] - S[i] + 1) \rceil$ bits by subtracting the lower bound $S[i]$ from their values. Plenty of variants of this simple approach has been proposed (Silvestri and Venturini 2010; Delbru et al. 2012; Lemire and Boytsov 2013). Recently, it has also been shown (Ottaviano et al. 2015) that using more than one compressor to represent the blocks, rather than simply representing all blocks with the same compressor, can introduce significant improvements in query time within the same space constraints.

Among the simplest binary packing strategies, Simple-9 and Simple-16 (Anh and Moffat 2005, 2010) combine very good compression ratio and high decompression speed. The key idea is to try to pack as many integers as possible in a memory word (32 or 64 bits). As an example, Simple-9 uses 4 *header* bits and 28 *data* bits. The header bits provide information on how many elements are packed in the data segment using equally sized code words. The 4 header bits distinguish from 9 possible configurations. Similarly, Simple-16 has 16 possible configurations.

PForDelta

The biggest limitation of block-based strategies is their space inefficiency whenever a block contains at least one large value, because this forces the compressor to encode *all* values with the number of bits needed to represent this large value (universe of representation). This has been the main motivation for the introduction of PForDelta (PFD) (Zukowski et al. 2006). The idea is to choose a proper value k for the universe of representation of the block, such that a large fraction, e.g., 90%, of its integers can be written in k bits each. This strategy is called *patching*. All integers that do not fit in k bits are treated as

exceptions and encoded separately using another compressor.

More precisely, two configurable parameters are chosen: a base value b and a universe of representation k so that most of the values fall in the range $[b, b + 2^k - 1]$ and can be encoded with k bits each by shifting (delta-encoding) them in the range $[0, 2^k - 1]$. To mark the presence of an exception, we also need a special *escape* symbol; thus we have $[0, 2^k - 2]$ possible configurations for the integers in the block.

The variant OptPFD (Yan et al. 2009), which selects for each block the values of b and k that minimize its space occupancy, has been demonstrated to be more space-efficient and only slightly slower than the original PFD (Yan et al. 2009; Lemire and Boytsov 2013).

Elias-Fano

The encoding we are about to describe was independently proposed by Elias (1974) and Fano (1971), hence its name. Given a monotonically increasing sequence $S(n, u)$, i.e., $S[i-1] \leq S[i]$, for any $1 \leq i < n$, with $S[n-1] < u$, we write each $S[i]$ in binary using $\lceil \log u \rceil$ bits. The binary representation of each integer is then split into two parts: a *low* part consisting in the rightmost $\ell = \lceil \log \frac{u}{n} \rceil$ bits that we call *low bits* and a *high* part consisting in the remaining $\lceil \log u \rceil - \ell$ bits that we similarly call *high bits*. Let us call ℓ_i and h_i the values of low and high bits of $S[i]$, respectively. The Elias-Fano representation of S is given by the encoding of the high and low parts.

The array $L = [\ell_0, \dots, \ell_{n-1}]$ is written explicitly in $n \lceil \log \frac{u}{n} \rceil$ bits and represents the encoding of the low parts. Concerning the high bits, we represent them in *negated unary* using a bit vector of $n + 2^{\lfloor \log n \rfloor} \leq 2n$ bits as follows. We start from a 0-valued bit vector H , and set the bit in position $h_i + i$, for all $i \in [0, n)$. Finally the Elias-Fano representation of S is given by the concatenation of H and L and overall takes $\text{EF}(S(n, u)) \leq n \lceil \log \frac{u}{n} \rceil + 2n$ bits.

While we can opt for an arbitrary split into high and low parts, ranging from 0 to $\lceil \log u \rceil$, it can be shown that $\ell = \lceil \log \frac{u}{n} \rceil$ minimizes the overall space occupancy of the encod-

ing (Elias 1974). Figure 1 shows an example of encoding for the sequence $S(12, 62) = [3, 4, 7, 13, 14, 15, 21, 25, 36, 38, 54, 62]$.

Despite its simplicity, it is possible to randomly access an integer from a sequence encoded with Elias-Fano *without* decompressing the whole sequence. We refer to this operation as **Access**(i), which returns $S[i]$. The operation is supported using an auxiliary data structure that is built on bit vector H , able to efficiently answer **Select**₁(i) queries, that return the position in H of the i -th 1 bit. This auxiliary data structure is *succinct* in the sense that it is negligibly small in asymptotic terms, compared to **EF**($S(n, u)$), requiring only $o(n)$ additional bits (Navarro and Mäkinen 2007; Vigna 2013).

Using the **Select**₁ primitive, it is possible to implement **Access**(i) in $O(1)$ time. We basically have to relink together the high and low bits of an integer, previously split up during the encoding phase. The low bits ℓ_i are trivial to retrieve as we need to read the range of bits $L[i\ell, (i+1)\ell)$. The retrieval of the high bits deserves, instead, a bit more care. Since we write in negated unary how many integers share the same high part, we have a bit set for every integer of S and a zero for every distinct high part. Therefore, to retrieve the high bits of the i -th integer, we need to know how many zeros are present in $H[0, \text{Select}_1(i)]$. This quantity is evaluated on H in $O(1)$ as **Rank**₀(**Select**₁(i)) = **Select**₁(i) - i . Finally, linking the high and low bits is as simple as: **Access**(i) = ((**Select**₁(i) - i) \ll ℓ) | ℓ_i , where \ll is the left shift operator and $|$ is the bitwise OR.

The query **NextGEQ**(x) is supported in $O(1 + \log \frac{u}{n})$ time as follows. Let h_x be the high bits of x . Then for $h_x > 0$, $i = \text{Select}_0(h_x) - h_x + 1$ indicates that there are i integers in S whose high bits are less than h_x . On the other hand, $j = \text{Select}_0(h_x + 1) - h_x$ gives us the position at which the elements having high bits greater than h_x start. The corner case $h_x = 0$ is handled by setting $i = 0$. These two preliminary operations take $O(1)$. Now we have to conclude our search in the range $S[i, j]$, having *skipped* a potentially large range

Inverted Index

Compression, Fig. 1 An example of Elias-Fano encoding. We distinguish in light gray the *missing high bits*, i.e., the ones not belonging to the integers of the sequence among the possible $2^{\lfloor \log n \rfloor}$

	3	4	7	13	14	15	21	25	36	38	54	62
high	0	0	0	0	0	0	0	0	1	1	1	1
	0	0	0	0	0	0	1	1	0	0	0	1
	0	0	0	1	1	1	0	1	0	0	1	0
low	0	1	1	1	1	1	1	0	1	1	1	1
	1	0	1	0	1	1	0	0	0	1	1	1
H	1	0	1	1	0	1	1	1	0	0	0	0
	1	1	1	0	1	1	1	1	1	0	10	10
	0	0	1	1	1	1	1	1	1	0	110	110
L	001100111	101110111	101	001	100110						110	110

of elements that, otherwise, would have required to be compared with x . We finally determine the successor of x by binary searching in this range which may contain up to u/n integers. The time bound follows.

Partitioned Elias-Fano

One of the most relevant characteristics of Elias-Fano is that it only depends on two parameters, i.e., the length and universe of the sequence. As inverted lists often present groups of close docIDs, Elias-Fano fails to exploit such natural clusters. Partitioning the sequence into chunks can better exploit such regions of close docIDs (Ottaviano and Venturini 2014).

The core idea is as follows. The sequence $S(n, u)$ is partitioned into n/b chunks, each of b integers. First level L is made up of the last elements of each chunk, i.e., $L = [S[b - 1], S[2b - 1], \dots, S[n - 1]]$. This level is encoded with Elias-Fano. The second level is represented by the chunks themselves, which can be again encoded with Elias-Fano. The main reason for introducing this two-level representation is that now the elements of the i -th chunk are encoded with a smaller universe, i.e., $L[i] - L[i - 1] - 1, i > 0$. As the problem of choosing the best partition is posed, an algorithm based on dynamic programming can be used to find in $O(n \log_{1+\epsilon} 1/\epsilon)$ time partition whose cost (i.e., the space taken by the partitioned encoded sequence) is at most $(1 + \epsilon)$ away from the optimal one, for any $0 < \epsilon < 1$ (Ottaviano and Venturini 2014).

This inverted list organization introduces a level of indirection when resolving NextGEQ queries. However, this indirection only causes a very small time overhead compared to plain

Elias-Fano on most of the queries (Ottaviano and Venturini 2014). On the other hand, partitioning the sequence greatly improves the space efficiency of its Elias-Fano representation.

Binary Interpolative

Binary Interpolative coding (BIC) (Moffat and Stuiver 2000) is another approach that, like Elias-Fano, directly compresses a monotonically increasing integer sequence without a first delta-encoding step. In short, BIC is a recursive algorithm that first encodes the middle element of the current range and then applies this encoding step to both halves. At each step of recursion, the algorithm knows the reduced ranges that will be used to write the middle elements in fewer bits during the next recursive calls.

More precisely, consider the range $S[i, j]$. The encoding step writes the quantity $S[m] - low - m + i$ using $\lceil \log(hi - low - j + i) \rceil$ bits, where $S[m]$ is the range middle element, i.e., the integer at position $m = (i + j)/2$, and low and hi are, respectively, the lower bound and upper bound of the range $S[i, j]$, i.e., two quantities such that $low \leq S[i]$ and $hi \geq S[j]$. The algorithm proceeds recursively, by applying the same encoding step to both halves: $[i, m]$ and $[m + 1, j]$ by setting $hi = S[m] - 1$ for the left half and $low = S[m] + 1$ for the right half. At the beginning, the encoding algorithm starts with $i = 0$, $j = n - 1$, $low = S[0]$, and $hi = S[n - 1]$. These quantities must be also known at the beginning of the decoding phase. Apart from the initial lower and upper bound, all the other values of low and hi are computed on the fly by the algorithm.

Inverted Index Compression, Fig. 2 An example of Binary Interpolative coding. We highlight in bold font the middle element currently encoded: above each element we report a pair where the first value indicates the element *actually* encoded and the second value the universe of representation

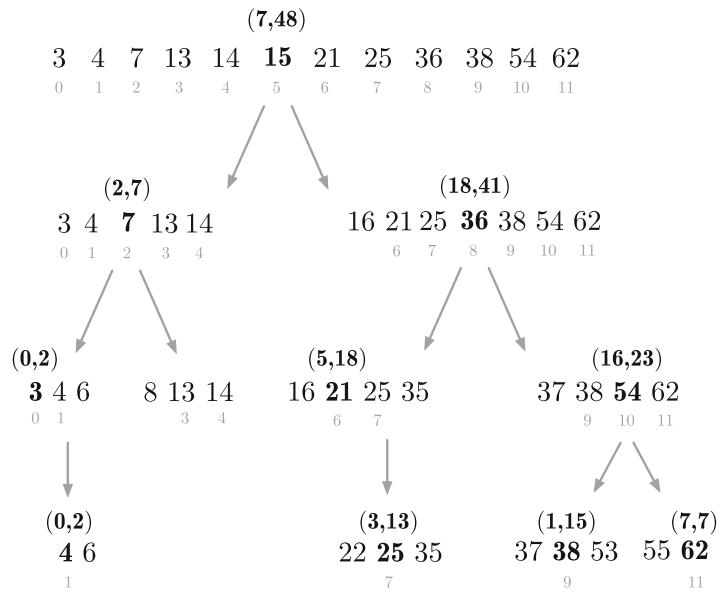


Figure 2 shows an encoding example for the same sequence of Fig. 1. We can interpret the encoding as a preorder visit of the binary tree formed by the recursive calls the algorithm performs. The encoded elements in the example are in order: [7, 2, 0, 0, 18, 5, 3, 16, 1, 7]. Moreover, notice that whenever the algorithm works on a range $S[i, j]$ of consecutive integers, such as the range $S[3, 4] = \{13, 14\}$ in the example, i.e., the ones for which the condition $S[j] - S[i] = j - i$ holds, it stops the recursion by emitting no bits at all. The condition is again detected at decoding time, and the algorithm implicitly decodes the run $S[i], S[i] + 1, S[i] + 2, \dots, S[j]$. This property makes BIC extremely space-efficient whenever the encoded sequences feature clusters of docIDs (Witten et al. 1999; Yan et al. 2009; Silvestri and Venturini 2010), which is the typical case for inverted lists. The key inefficiency of BIC is, however, the decoding speed which is highly affected by the recursive nature of the algorithm. This is even more evident considering random access to individual integers that cannot be performed in constant time as it is the case for Elias-Fano (see 1).

Future Directions for Research

This concluding section is devoted to recent approaches that encode many inverted lists together to obtain higher compression ratios. This is possible because the inverted index naturally presents some amount of redundancy, caused by the fact that many docIDs are shared between its lists. In fact, as already motivated at the beginning of section “List Compressors”, the identifiers of similar documents will be stored in the inverted lists of the terms they share.

Pibiri and Venturini (2017) proposed a clustered index representation. The inverted lists are divided into clusters of similar lists, i.e., the ones sharing as many docIDs as possible. Then for each cluster, a *reference list* is synthesized with respect to which all lists in the cluster are encoded. In particular, the *intersection* between the cluster reference list and a cluster list is encoded more efficiently: all the docIDs are implicitly represented as the *positions* they occupy within the reference list. This makes a big improvement for the cluster space, since each intersection can be rewritten in a much smaller universe. Although

any list compressor supporting NextGEQ can be used to represent the (rank-encoded) intersection and the residual segment of each list, the paper adopted partitioned Elias-Fano (see 1). With an extensive experimental analysis, the proposed clustered representation is shown to be superior to partitioned Elias-Fano and also Binary Interpolative coding for index space usage. By varying the size of the reference lists, different time/space trade-offs can be obtained.

Reducing the redundancy in highly repetitive collections has also been advocated in the work by Claude, Fariña, Martínez-Prieto, and Navarro (2016). The described approach first transforms the inverted lists into lists of d -gaps and then applies a general, i.e., *universal*, compression algorithm to the sequence formed by the concatenation of all the lists. The compressed representation is also enriched with pointers to mark where each individual list begins. The paper experiments with Re-Pair compression (Larsson and Moffat 1999), VByte, and LZMA (<http://www.7-zip.org/>) that is an improved version of the classic LZ77 (Ziv and Lempel 1977) and run-length encoding. The experimental analysis reveals that significant reduction in space is possible on highly repetitive collections, e.g., versions of Wikipedia, with respect to encoders tailored for inverted indexes while only introducing moderate slowdowns.

An approach based on generating a *context-free grammar* from the inverted index has been proposed by Zhang, Tong, Huang, Liang, Li, Stones, Wang, and Liu (2016). The core idea is to identify common patterns, i.e., repeated subsequences of docIDs, and substitute them with symbols belonging to the generated grammar. Although the reorganized inverted lists and grammar can be suitable for different encoding schemes, the authors adopted OptPFD (Yan et al. 2009). The experimental analysis indicates that good space reductions are possible compared to state-of-the-art encoding strategies with competitive query processing performance. By exploiting the fact that the identified common patterns can be placed directly in the final result

set, the query processing performance can also be improved.

Cross-References

- Grammar-Based Compression
- (Web/Social) Graph Compression

References

- Anh VN, Moffat A (2005) Inverted index compression using word-aligned binary codes. *Inf Retr J* 8(1):151–166
- Anh VN, Moffat A (2010) Index compression using 64-bit words. *Softw Pract Exp* 40(2):131–147
- Claude F, Fariña A, Martínez-Prieto MA, Navarro G (2016) Universal indexes for highly repetitive document collections. *Inf Syst* 61:1–23
- Dean J (2009) Challenges in building large-scale information retrieval systems: invited talk. In: Proceedings of the 2nd international conference on web search and data mining (WSDM)
- Delbru R, Campinas S, Tummarello G (2012) Searching web data: an entity retrieval and high-performance indexing model. *J Web Semant* 10:33–58
- Dhulipala L, Kabiljo I, Karrer B, Ottaviano G, Pupyrev S, Shalita A (2016) Compressing graphs and indexes with recursive graph bisection. In: Proceedings of the 22nd international conference on knowledge discovery and data mining (SIGKDD), pp 1535–1544
- Elias P (1974) Efficient storage and retrieval by content and address of static files. *J ACM* 21(2):246–260
- Elias P (1975) Universal codeword sets and representations of the integers. *IEEE Trans Inf Theory* 21(2):194–203
- Fano RM (1971) On the number of bits required to implement an associative memory. Memorandum 61. Computer Structures Group, MIT, Cambridge
- Goldstein J, Ramakrishnan R, Shaft U (1998) Compressing relations and indexes. In: Proceedings of the 14th international conference on data engineering (ICDE), pp 370–379
- Golomb S (1966) Run-length encodings. *IEEE Trans Inf Theory* 12(3):399–401
- Larsson NJ, Moffat A (1999) Offline dictionary-based compression. In: Data compression conference (DCC), pp 296–305
- Lemire D, Boytsov L (2013) Decoding billions of integers per second through vectorization. *Softw Pract Exp* 45(1):1–29
- Lemire D, Kurz N, Rupp C (2018) Stream-VByte: faster byte-oriented integer compression. *Inf Process Lett* 130:1–6

- Moffat A, Stuiver L (2000) Binary interpolative coding for effective index compression. *Inf Retr J* 3(1): 25–47
- Navarro G, Mäkinen V (2007) Compressed full-text indexes. *ACM Comput Surv* 39(1):1–79
- Ottaviano G, Venturini R (2014) Partitioned elias-fano indexes. In: Proceedings of the 37th international conference on research and development in information retrieval (SIGIR), pp 273–282
- Ottaviano G, Tonello N, Venturini R (2015) Optimal space-time tradeoffs for inverted indexes. In: Proceedings of the 8th annual international ACM conference on web search and data mining (WSDM), pp 47–56
- Pibiri GE, Venturini R (2017) Clustered Elias-Fano indexes. *ACM Trans Inf Syst* 36(1):1–33. ISSN 1046-8188
- Plaisance J, Kurz N, Lemire D (2015) Vectorized VByte decoding. In: International symposium on web algorithms (iSWAG)
- Rice R, Plaunt J (1971) Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Trans Commun* 16(9):889–897
- Salomon D (2007) Variable-length codes for data compression. Springer, London
- Silvestri F (2007) Sorting out the document identifier assignment problem. In: Proceedings of the 29th European conference on IR research (ECIR), pp 101–112
- Silvestri F, Venturini R (2010) Vsencoding: efficient coding and fast decoding of integer lists via dynamic programming. In: Proceedings of the 19th international conference on information and knowledge management (CIKM), pp 1219–1228
- Stepanov A, Gangolli A, Rose D, Ernst R, Oberoi P (2011) Simd-based decoding of posting lists. In: Proceedings of the 20th international conference on information and knowledge management (CIKM), pp 317–326
- Vigna S (2013) Quasi-succinct indices. In: Proceedings of the 6th ACM international conference on web search and data mining (WSDM), pp 83–92
- Witten I, Moffat A, Bell T (1999) Managing gigabytes: compressing and indexing documents and images, 2nd edn. Morgan Kaufmann, San Francisco
- Yan H, Ding S, Suel T (2009) Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th international conference on world wide web (WWW), pp 401–410
- Zhang Z, Tong J, Huang H, Liang J, Li T, Stones RJ, Wang G, Liu X (2016) Leveraging context-free grammar for efficient inverted index compression. In: Proceedings of the 39th international conference on research and development in information retrieval (SIGIR), pp 275–284
- Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. *IEEE Trans Inf Theory* 23(3):337–343
- Zukowski M, Héman S, Nes N, Boncz P (2006) Super-scalar RAM-CPU cache compression. In: Proceedings of the 22nd international conference on data engineering (ICDE), pp 59–70

Isolation Level

- Database Consistency Models

IT Security

- Big Data for Cybersecurity

Iterative Refinement

- Tabular Computation

J

Job Scheduling

- ▶ [Advancements in YARN Resource Manager](#)
-

Julia

Zacharias Voulgaris

Data Science Partnership Ltd, London, UK
Thinkful LLC, New York, USA
Technics Publications LLC, Basking Ridge,
NJ, USA

Overview of the Julia Project and Historical Context

The inception of Julia was by Jeff Bezanson and his adviser, Prof. Alan Edelman, when the former was doing his PhD thesis at MIT. However, the project soon gathered momentum that went beyond that particular thesis encouraging more individuals to join it, contributing to it in various ways. Soon, Stefan Karpinski and Viral B. Shah undertook key roles in the project and turned this into a group effort.

The idea behind Julia was to have a programming language that is high-level (and therefore easy to prototype in), while at the same time exhibit high performance, comparable to that of low-level languages, such as C and Fortran.

Such a programming language would solve the two-language problem, which involves creating a script in a high-level language in order to prove a concept and then translating it into a low-level language to put that program in production. Even at its first stages, Julia managed to do that, though the lack of libraries (packages) made it more of a niche language that would appeal mainly to those dedicated enough to create a lot of processes from scratch.

By 2012, however, when Julia became accessible to the public, there were enough packages to allow for some basic processes to be implemented in the language. As more and more people became aware of its potential and as benchmarking experiments on various applications were conducted, the performance edge of Julia was beyond any reasonable doubt. Naturally, this encouraged many programmers to take it up and gradually build packages for it, most of which were made available through its website ([Official Julia Language 2017](#)).

Soon after that, Julia became an appealing tool both for businesses and universities alike. Apart from MIT, there were a few other universities around the world (including one in Europe) that taught Julia as part of a technical course. With the version 1.0 of the language still pending, this was a noteworthy accomplishment, especially considering that there were noticeable changes from one release to the next, rendering compatibility issues in many of the packages and scripts that were developed for earlier versions of the language.

Currently, Julia has packages for the vast majority of data science applications, including some niche ones, like T-SNE, a stochastic dimensionality reduction technique developed by Prof. Laurens van der Maaten in the Netherlands. Also, despite the common misconception that Julia is not data science ready yet, packages for all common data science applications exist in Julia right now, while anything that isn't there can be implemented without any compromise in performance. This is because a large part of the Julia code base is written in Julia itself (Official Julia Language 2017).

Key Characteristics of the Julia Language

There are various characteristics of the language that make it stand out from other programming languages and which are often used to promote it as a data science technology. These include (but are not limited to) the following (Voulgaris 2017):

- **Prototyping speed.** It is very easy to create a proof-of-concept script in Julia, implementing a data analytics algorithm or an ETL process.
- **Performance speed.** Scripts written in Julia are very fast to execute, be it on a single CPU or in parallel. This renders Julia ideal for production code, particularly for computationally expensive applications.
- **Easy to learn.** Although many programming languages require a certain familiarity with programming, Julia is fairly straightforward right off the shelf. Also, if someone has coded in Python, R, or MATLAB, it will seem even easier to learn.
- **Versatile in terms of hardware.** Julia works well in all major OSes, across various hardware configurations (including tiny computers).
- **Compatibility with other programming languages.** Julia scripts can be called from other languages, like Python and R, while

you can call scripts from other languages (including C), via the Julia kernel.

- **Growing user community.** Although the number of users of Julia is not as large as that of the users of other languages, there are communities all over the world through meetup, as well as some strictly online ones. Also, there are conferences on Julia taking place regularly.
- **Built-in package manager.** As the use of packages is commonplace in all high-level languages, Julia was designed with this in mind. As a result, it is possible to add, remove, and update packages (libraries) of the language within its kernel, using some simple commands.
- **Unicode support (not limited to UTF-8 only).** Julia can handle a variety of characters, including letters from non-Latin alphabets and special mathematical and scientific symbols. This renders the processing of text much more efficient and with a broader scope.
- **Multiple dispatch.** This is a very useful feature of the language enabling the programmer to have functions with the same name but taking different inputs. The idea is that the user of these functions doesn't have to remember numerous names for specialized versions of the same function, as Julia itself decides which one to use, based on the inputs it is given. Multiple dispatch therefore allows for cleaner code and easier maintenance of scripts, enabling the user to leverage the functional programming paradigm, saving time and effort.

Key Julia Packages for Data Science

Beyond the base package of Julia, there are several packages out there which can be used for data science-related applications. The vast majority of these packages are available through the Julia package repository at the official Julia site. This repository contains all official Julia packages, which have undergone unit testing and have been tried on various benchmarks to ensure

they produce the expected results. However, reliable Julia packages can be found in other sites too, since there are various GitHub repositories containing nonofficial packages (e.g., the package for the T-SNE method, from its creator's repository).

The key packages that are most relevant for data science include the following:

1. **StatsBase** – a collection of various statistics functions, essential in most data science projects
2. **HypothesisTests** – a package containing several statistical methods for hypothesis testing (e.g., t-tests, chi-square tests, etc.)
3. **Gadfly** – one of the best plotting packages out there, written entirely in Julia
4. **PyPlot** – a great plotting package, borrowed from Python's Matplotlib; ideal for heat maps, among other plot types
5. **Clustering** – a package specializing in clustering methods
6. **DecisionTree** – decision trees and random forests package
7. **Graphs** – a graph analytics package (the most complete one out there)
8. **LightGraphs** – a package with relatively fast graph algorithms (not compatible with Graphs package)
9. **DataFrames** – the Julia equivalent of Python's pandas package
10. **Distances** – a very useful package for distance calculation, covering all major distance metrics
11. **GLM** – generalized linear model package for regression analysis
12. **T-SNE** – a package implementing the T-SNE algorithm by the creators of the algorithm
13. **ELM** – the go-to package for extreme learning machines, having a single hidden layer
14. **MultivariateStats** – a great place to obtain various useful statistics functions, including principal components analysis (PCA)
15. **MLBase** – an excellent resource for various support functions for machine learning applications
16. **MXNet** – the Julia API for Apache's MXNet, a great option for deep-learning applications

Even though this list is not exhaustive, it covers the majority of data science applications. Also, some applications may have more than one package out there. The packages here are the ones found to be better maintained, at the time of this writing. All of them, apart from T-SNE, can be found in the Julia package repository.

Implementing and Running Custom Algorithms in Julia

Let's take a look at how you can create a script in Julia, implementing a custom algorithm. For this example, we'll use the k nearest neighbor (kNN) algorithm for classification, which is one of the simplest machine learning systems out there. Although there is a package in Julia that provides a solid implementation of kNN, we'll examine how you can develop a similar script from scratch.

For starters, we'll need a distance function that calculates the Euclidean distance between two points, x and y . This is an auxiliary function, as it is not part of the actual kNN algorithm. Also, even though functions like this exist in the Distances package, it's useful to see how you can implement it here too, to see how you can use different functions in a script. A typical implementation of the Euclidean distance is the following:

```
function distance{T<:Number}(x::Array{T,1}, y::Array{T,1})
    dist = 0
    for i in 1:length(x)
        dist += (x[i] - y[i])^2
    end
    dist = sqrt(dist)
    return dist
end
```

An essential part of the kNN classification algorithm is the processing of a given element whose class is unknown, given the class information in the known elements. To accomplish this, it takes the k closest points (nearest neighbors) to that unknown point and performs a majority vote decision-making process, to figure out which class it should belong to. Despite its simplicity, this process is actually quite effective, given the

right selection of the k parameter. Below is a typical implementation of that part of the algorithm as an independent function.

```
function classify{T<:Any}
    (distances::Array{Float64, 1},
     labels::Array{T, 1, k::Int64})
        class = unique(labels)
        nc = length(class)
        indexes = Array(Int, 0)
        M = maxtype(typeof(distances[1]))
        class_count = Array(Int, nc)
        for i in 1:k
            indexes[i] = inmin(distances)
            distances[indexes[i]] = M
        end
        klables = labels[indexes]

        for i in 1:nc
            for j in 1:k
                if klables[j]
                    == class[i]
                    class_count[i]
                    += 1
                    break
                end
            end
        end
        index = inmax(class_count)
        return class[index]
    end
```

So, what would the actual kNN algorithm look like if it were implemented using the above functions as auxiliaries? The short function that follows is a typical implementation of the wrapper function of the algorithm, which makes use of `distance()` and `classify()` in order to make a series of predictions, for the classes of the data points in matrix Y . As expected, the number of neighbors k is passed as a parameter, just like in the `classify()` function.

```
function apply_kNN{T1<:Number, T2<:Any}
    (X::Array{T1,2}, x::Array{T2,1},
     Y::Array{T1,2}, k::Int)
    N = size(X,1)
    n = size(Y,1)
    D = Array(Float64, N)
    z = Array(typeof(x[1]), n)
    for i in 1:n
        for j in 1:N
            D[j] = distance(X[j,:], Y[i,:])
        end
        z[i] = classify(D, x, k)
    end
```

```
    end
    return z
end
```

This example was taken from Voulgaris (2016), where various aspects of the Julia language for data science applications are explored.

Structuring Code and Performance Measurement of Julia Scripts

Just like most programming languages, Julia has a set of built-in functions for measuring the performance of pieces of code written and executed in it. This is useful not only for gauging how much time it takes to run a script but also for figuring out how much memory the script requires, something particularly useful for scripts you wish to put to scale, for tackling more challenging data science problems.

First of all, let's start by defining what a chunk of code is for Julia. This is a set of commands in some kind of programming structure, such as a for loop or a script in a .jl file. However, for assessing individual parts of a program, you may want to use a block of code that is neither one of these forms. In this case, you can use a custom code block in the form of:

```
begin
    command 1
    command 2
    .
    .
    command n
end
```

Note that the indentation in the commands between “begin” and “end” is entirely optional. This is because Julia doesn't care about how you format the code, as long as it's clear where one command ends and another one begins. Instead of the line break, for example, you could always use the semi-colon (;) which is also used to suppress the output of a command when run. So, a block of code using the aforementioned structure could take the form:

```
begin; command 1; command 2; ...
    command n; end
```

This is quite useful if you wish to try something out to see if it works, while you are on the kernel window. Also, if you want to make your code obscure so that people will be more hesitant to copy it, though this tactic may not deter the more determined ones. Moreover, this compact way of writing code is useful when you want to measure its performance, particularly if you are on the kernel window.

Measuring the performance of a piece of code in Julia is fairly straightforward. The functions most commonly used are @time, @elapsed, and @allocated. Note the “@” is part of their name. This is a special character indicating that the function is a meta-command, something that applies not on regular data but on code instead. Meta-commands are part of meta-programming, which is a more advanced topic, beyond the scope of this article.

So, if we have a piece of code (e.g., begin; x = 0; for i = 1:2:1000000; x += i; end; println(x); end, which basically adds up all the odd integers between 1 and 1000000), we can gauge its performance by running the following code:

```
@time begin; x = 0;
for i = 1:2:1000000;
x += i; end; println(x); end
```

The output of this would be twofold. First, the actual result of the code will be given (from the println() command, which outputs something on the screen, adding a line break at the end of it). Following that, there will be three numbers in a single line. The first one will be the number of seconds it took Julia to run the script. Following that, there are a couple of numbers inside parentheses related to the memory usage of that code. The first of them has to do with the number of memory allocations performed, while the latter, with the actual memory usage in bytes, MiBs, etc. In some cases, there is a third number in the parenthesis, denoting the proportion of the time of that code that is used in the language’s built-in garbage collector (usually referred to as “gc”). Note that because of the gc and other reasons, when you first run the script, it always takes slightly longer than any consecutive runs, while the memory allocation is also higher then.

The @elapsed meta-command will just yield the number of seconds, and it is an actual number, of Float64 type to be exact (or Float32 if you are using a 32-bit processor). The use case of this function is storing and later on processing the times of a particular script (e.g., a data science model), for benchmarking purposes.

Similar to @elapsed is the @allocated meta-command. This basically returns an Int64 number reflecting the number of memory allocations of the given code. However, this may not include all of the memory allocations, so it is bound to differ from the number given in the output of the @time meta-command.

There are additional tools for evaluating the performance of a piece of code in Julia. Namely, if you are particularly interested in benchmarking, you will find the BenchmarkingTools package (benchmarkingtools.jl) by Jarrett Revels quite useful. In that package you can explore various options for more detailed benchmarking, such as the @benchmark meta-command, which performs a series of tests and provides some descriptive statistics of the results, so that you get a more in-depth understanding of the resources required in your code.

J

Firing Up Additional CPUs Through the Julia Kernel

It is often the case that we need additional CPUs (or GPUs) in order to carry out some tasks that are too computationally expensive for a single processor to handle. This is a usually advanced topic in data science, and it is usually used in deep-learning applications. However, in Julia it is fairly straightforward, and it can be done with a small set of commands that are in the base package of the language.

The process for performing parallelization is the following:

1. Add new processors to Julia (i.e., give Julia access to your computing resources, aka processors or threads).

2. Create one or more mapper functions that will be run on each processor, and make them available to all of the processors.
3. Create a wrapper function that will also act as a reducer or aggregator.
4. Test each mapper to make sure that it provides the desired result.
5. Run the wrapper function on two or so processors to ensure that it works as expected.
6. Make any changes if necessary.
7. Run the wrapper function on all of your processors.

For all of this, the following commands are essential:

- **nprocs()**: yields the number of processors available through Julia.
- **addprocs(n)**: adds n processors to Julia (n is an integer). The output of this function is an array containing the numbers of the added processors. Alternatively, you can give Julia access to n processors when firing it up by typing in the command line: `julia -p n`.
- **procs()**: yields an array of all the available processors (i.e., their code numbers).
- **@parallel**: performs a certain command, usually a loop, in parallel (note that indexing is not maintained in this case, so this command is useful for simulations mainly).
- **@everywhere**: makes the function that follows available in all the processors accessible to Julia. This is essential for parallelizing a custom function. If this meta-command is omitted, the functions you define will be accessible only to the active CPU.
- **pmap()**: this function is nonequivalent to `map()`, but instead of mapping a function to the various elements of an array, it does so across all available processors. The equivalent function in R is `sapply()`.

It is important to keep in mind that adding more CPUs (workers) has an inherent overhead. So, unless the process you want to parallelize is complex enough, the benefit may not be as meaningful. In other words, before adding more Julia

instances on the various CPUs/GPUs, first make sure that the process is framed in an efficient way and that it is computationally heavy enough to render this kind of approach.

Other Aspects of Working with Julia

Working with Julia is straightforward, considering that it is a fairly new language. For those who prefer an interactive prompt, similar to IPython, there is the Julia kernel, which comes with the installation of the language.

If you prefer a “cleaner” programming environment, there is also the option of Jupyter, a popular IDE among Python users. Jupyter is fully compatible with Julia, and it is the preferred option for many data scientists. One thing to keep in mind is that for Jupyter and Julia to work together, you need to have installed the IJulia package, as well as the anaconda package in Python.

However, even a basic text editor could work well with Julia, since there are editors like Notepad++ and Atom, which understand Julia code and enable useful coloring of certain keywords. Some Julia programmers also make use of Juno, which is a text editor that is linked to the Julia kernel, so you can execute specific parts of your code, without leaving the editor. Juno is based on the Atom editor, and it is distributed as part of a software bundle, from the official Julia site.

Julia Resources

Beyond the short introduction of the Julia language in this article, there are several other resources for the language that are recommended for more in-depth understanding of its functionality and usefulness, in a data science setting. Namely, the Julia wiki book ([2017](#)) is an excellent resource for the various technical aspects of the language, with many examples of how they are applied in practice. Also, the *Julia for Data Science* book (Technics Publications) (Voulgaris

2016) is a great place to start for learning not only how Julia can be applied in data science but also about the data science field itself.

Finally, as Julia is constantly evolving, it would be a good idea to follow its developments through the various blog articles written for it, through the Julia Bloggers meta-blog, an aggregate of the various blogs subscribed to this site. Not all of the entries are relevant for someone learning Julia, but all of them are useful for gaining a better understanding of the language's usefulness and applications.

References

- Julia wiki book. https://en.wikibooks.org/wiki/Introducing_Julia. Last accessed 3 Oct 2017
- Official Julia Language Website: www.julialang.org. Last accessed 30 Sept 2017
- Parallelization presentation by Dr. Jeff Bezanson. <https://www.youtube.com/watch?v=JoRn4ryMcIc>. Last accessed April 2018
- Voulgaris Z (2016) Julia for data science. Technics Publications, Basking Ridge, NJ 07920
- Voulgaris Z (2017) Data science mindset, methodologies, and misconceptions. Technics Publications, Basking Ridge, NJ 07920

K

Key-Value Store

► NoSQL Database Systems

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption

Peng Jiang¹, Fuchun Guo², Willy Susilo², and Jinguang Han³

¹Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong

²Institute of Cybersecurity and Cryptology, School of Computing and Information

Technology, University of Wollongong, Wollongong, NSW, Australia

³Department of Computer Science, Surrey Centre for Cyber Security, University of Surrey, Guildford, UK

Definitions

Keyword privacy, Keyword-guessing attacks, Searchable encryption, Public key.

Overview

The notion of public key encryption with keyword search (PEKS) provides a flexible and

efficient approach to retrieve encrypted data stored in a remote server. A main threat in PEKS is the keyword guessing attack, where disclosed keywords leak the associated data and the user's interest. This gives rise to that the keyword privacy becomes a concern in searchable encryption. In this paper, we present a brief history of keyword privacy in searchable encryption. We make a systematic investigation of keyword privacy issue in public key-based searchable encryption. We describe keyword guessing attacks in a general PEKS scheme. According to different attackers, we refine inherent reasons and summarize enhancement achievements. This preserves the keyword privacy and provides guidance for the secure development of searchable encryption.

Introduction

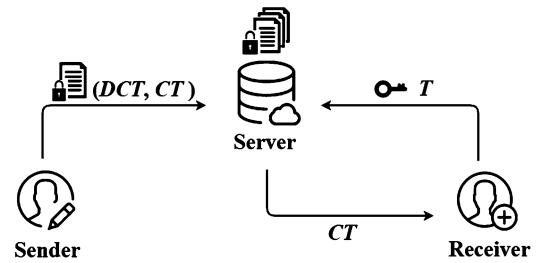
Driven by economic benefits, cloud-based database systems have been widely adopted in data storage. This, on the one hand, brings profits to the service provider and, on the other hand, reduces the local storage cost for users. In such an outsourcing storage, the data owner will lose control of his/her data, and the cloud server is not fully trusted, especially in the public cloud, which pose that several security and privacy issues arise. Encryption is a fundamental technique in order to protect outsourced data from disclosure. However, an inherent problem is how to efficiently search the intended data. The

traditional plaintext-based search is unavailable for encrypted data. There are two straightforward solutions. One is that users download and decrypt all the encrypted data to find the target locally, and the other is that the server with the key from the user decrypts all the data to find and return the target to the user. These two solutions are either impractical for extremely large bandwidth or insecure for compromising the data privacy. How to search over encrypted data efficiently and securely becomes an important and interesting problem.

Searchable encryption (SE) is a promising tool to retrieve encrypted data and gains growing interests. It follows a sender-receiver-server mode, and the function of each party is described as follows.

- *Sender* owns the data associated with some keyword. He/she encrypts both the data and the keyword to generate the data ciphertext DCT and the searchable ciphertext CT , respectively. Then the sender uploads (DCT, CT) to the server.
- *Receiver* generates the trapdoor T with the keyword and his/her secret key to search the data associated with this keyword. Then the receiver submits T to the server for searching.
- *Server* provides data storage and data searching service. When receiving a trapdoor T , the server checks whether T matches some searchable ciphertext CT by running a test algorithm. It returns the corresponding data ciphertext DCT to the receiver if and only if the matching succeeds. Note that the server is not fully trusted, which means that it honestly executes the searching operation but curiously learns the data as well as its associated information.

In SE, both the searchable ciphertext and the trapdoor include the encrypted keyword. If a searchable ciphertext matches with the given trapdoor, it means that the keywords are identical and the search succeeds. Figure 1 depicts an overview of the SE architecture.



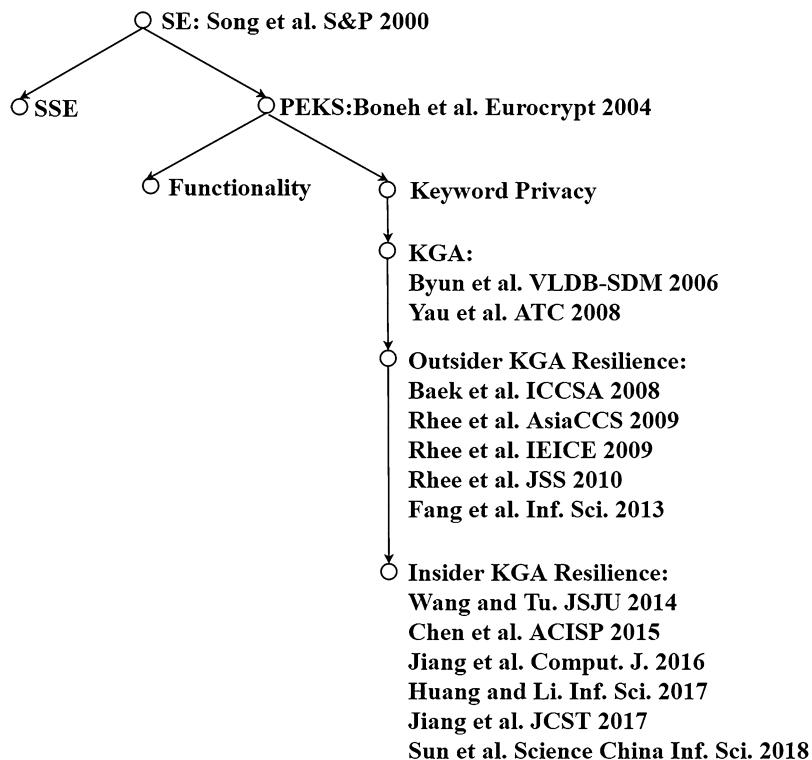
Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 1 SE architecture

SE mainly consists of two types of techniques, namely, symmetric searchable encryption (SSE) using symmetric cryptography and public-key encryption with keyword search (PEKS) using public-key cryptography.

- SSE was first introduced by Song et al. (2000). In SSE, both the searchable ciphertext and the trapdoor are generated with the same secret key. SSE enjoys a high searching performance (Kim et al. 2017; Bost et al. 2017; Kamara and Moataz 2017). However, SSE is subjected to complicated key distribution problems when data can be uploaded and searched in multiuser setting.
- PEKS was first introduced by Boneh et al. (2004). In PEKS, any senders can upload the data to the receiver using his/her public key instead of a shared key, and the receiver who has the corresponding secret key can search. PEKS and its variants (e.g., Boneh and Waters 2007; Sun et al. 2016; Jiang et al. 2016b) provide a flexible way to share data for senders and search data for receivers. PEKS does not need the key distribution due to the public-key cryptography and thus overcomes the weaknesses of SSE.

Contribution. In this entry, we present a brief history of PEKS in Fig. 2. We conduct an in-depth study of the keyword privacy in PEKS and its variants. Our study spans both keyword-guessing attacks and the corresponding defenses in terms of different attackers. Finally, we list

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 2 Brief history of PEKS



several future directions to stir up efforts into searchable encryption.

(Boneh and Franklin 2001). In comparison with SSE, PEKS has the following advantages.

- PEKS allows any senders to generate the searchable ciphertext with the public key of the receiver, while SSE allows only the key holder to generate the ciphertext. PEKS benefits the data sharing from any sender without a shared key.
- PEKS does not need a complicated key distribution when multiple senders/receivers are involved, but directly specifying in searchable ciphertext. PEKS has cost efficiency in key management.

An application example of PEKS as mentioned in Boneh et al. (2004) is the email system. Bob sends an encrypted email to Alice under Alice's public key, and the email server can test whether the email contains some keyword so as to route the email accordingly. If SSE is applied, Bob must contact Alice to obtain the

Public-Key Encryption with Keyword Search

Table 1 summarizes the major notations used in this entry.

The general PEKS algorithm is shown in Fig. 3. A brief description is as follows. The sender generates the searchable ciphertext using the keyword and the public key of a receiver, and the receiver who has the corresponding secret key produces the trapdoor using the keyword and his/her secret key. The server returns the corresponding data ciphertext to the receiver when the received trapdoor matches the searchable ciphertext. The PEKS construction proposed in Boneh et al. (2004) was derived from the technique of identity-based encryption (IBE)

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Table 1 Major notations used in this entry

Notation	Description
λ	Security parameter
pk_R/sk_R	Public/secret key of the receiver
pk_S/sk_S	Public/secret key of the server
pk_{FS}/sk_{FS}	Public/secret key of the front server
pk_{BS}/sk_{BS}	Public/secret key of the back server
pk_E/sk_E	Public/private key of the sender
w	Keyword
W	Keyword space
CT	Searchable ciphertext
T	Trapdoor
C_{ITS}	Internal testing state

- 1: $\text{KeyGen}(\lambda) \rightarrow (pk_R, sk_R)$
- 2: $\text{PEKS}(pk_R, w) \rightarrow CT$
- 3: $\text{Trapdoor}(sk_R, w) \rightarrow T$
- 4: $\text{Test}(pk_R, CT, T) \rightarrow 1/0$

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 3 PEKS algorithms

shared key before sending an encrypted email to Alice. Therefore, SSE becomes impractical in this scenario.

To improve PEKS, kinds of variants have been proposed. For example, public-key encryption with conjunctive keyword search (PECKS) schemes (Park et al. 2004; Boneh and Waters 2007; Bethencourt et al. 2006; Yang and Ma 2016) were proposed to strengthen the query expressiveness and support combinable multi-keyword search. Authorized keyword search schemes (Zheng et al. 2014; Shi et al. 2014; Sun et al. 2016; Jiang et al. 2016b, 2017b) were proposed to control the search via the user's attributes or the assigned keyword class.

Keyword-Guessing Attacks

A main concern in PEKS is the keyword privacy. The keyword space is usually low-entropy, instead of super-polynomially large. Anyone can

Algorithm 1 KGA algorithms

-
- 1: Given T for some unknown $w \in W$
 - 2: **for** $w' \in W$ **do**
 - 3: $CT \leftarrow \text{PEKS}(pk_R, w')$
 - 4: **if** $\text{Test}(pk_R, CT, T) = 1$ **then**
 - 5: $w = w'$
 - 6: **if** $w \neq w'$ and change another keyword
 - 7: Find the correct w
-

choose a keyword from the keyword space and generate a searchable ciphertext. The attacker always attempts to recover the keyword from a given trapdoor by checking whether his/her generated searchable ciphertext matches this trapdoor. If they match, the attack succeeds and finds the correct keyword; otherwise, the attacker continues to guess another keyword. This is called the keyword-guessing attack (KGA). PEKS (Boneh et al. 2004) was pointed out to be not secure against KGA (Byun et al. 2008; Yau et al. 2008).

We assume that there is a keyword set W in the system, where the size of W is N . Usually, the public key of any party is fully public and available to the whole system. KGA works as shown in Algorithm 1, where three steps includes “given a trapdoor,” “generate a searchable ciphertext,” and “execute the test.” The reason is that anyone who knows the public key of the receiver can generate the searchable ciphertext CT of arbitrary keyword himself. He/she then tries to check whether the generated CT matches the given trapdoor. If yes, the keyword in the trapdoor is identical to the keyword selected by the CT generator.

As the keyword always indicates the privacy of the user and his/her own data, it is therefore of practical importance to circumvent this vulnerability for searchable encryption. If the attacker is neither the server nor the receiver (i.e., outsider), a secure channel between the server and the receiver is an alternative solution. However, this compromises practicality since the deployment cost is quite expensive and such a system cannot use a regular public channel, such as Wi-Fi or 4G. If the attacker is the server itself (i.e., insider), the secure channel does not work, and hence some feasible solutions are desirable.

Privacy Enhancements Against KGA

KGA from outsider and insider are called *outsider attacks* and *insider attacks*, respectively. Depending on these two types of attacks, we summarize enhancement achievements for PEKS, which improve the keyword privacy.

Outsider Attacks Resilience

The essential reason for outsider attacks is that any outsider can run the test. To resist outsider attacks, the designated-tester PEKS was proposed (Rhee et al. 2009a) to restrict only the designated server to run the test. Figure 4 describes a general designated-tester PEKS construction. Its core idea is to introduce the public/private key pair into PEKS. Both the ciphertext and the trapdoor are generated with the public key of the server, such that only the designated server who holds the corresponding private key can do the test. The security model for designated-tester PEKS and the notion of trapdoor indistinguishability were defined in Rhee et al. (2009a, 2010), respectively. Designated-tester PEKS schemes proposed in Rhee et al. (2009b) and Fang et al. (2013) were provably secure in the random oracle model and in the standard model, respectively.

Insider Attacks Resilience

The technique used in outsider attacks resilience cannot work against insider attacks since the server has the secret key and is still able to do the test as Algorithm 1. There are two main reasons. One is that the ciphertext generation is public and the other is that the test is stand-alone.

- 1: $\text{Setup}(\lambda) \rightarrow SP$
- 2: $\text{KeyGen}_{\text{Server}}(SP) \rightarrow (pk_S, sk_S)$
- 3: $\text{KeyGen}_{\text{Receiver}}(SP) \rightarrow (pk_R, sk_R)$
- 4: $\text{PEKS}(SP, pk_R, pk_S, w) \rightarrow CT$
- 5: $\text{Trapdoor}(sk_R, pk_S, w) \rightarrow T$
- 6: $\text{Test}(SP, CT, T, sk_S) \rightarrow 1/0$

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 4 General construction against outsider attacks

Starting from these two reasons, two kinds of new frameworks for PEKS have been accordingly proposed to solve insider attacks.

The first framework is to introduce an authentication of the sender (e.g., the private key of the sender) into the searchable ciphertext, such that the server cannot generate a valid searchable ciphertext. A general authentication-based construction against insider attacks is shown in Fig. 5. Several constructions under this framework (Jiang et al. 2016a, 2017a; Huang and Li 2017; Sun et al. 2018) have been proposed. The online/offline keyword search scheme (Jiang et al. 2016b) achieved the quick generation of the trapdoor. The public-key authenticated encryption with keyword search scheme (Huang and Li 2017) was proved to be secure based on static assumptions. The constant-size trapdoor scheme (Jiang et al. 2017a) allowed to search ciphertexts from multiple senders with one trapdoor. The indistinguishability obfuscator-based scheme (Sun et al. 2018) considered the security against both the honest-but-curious server and the malicious server.

The second framework is to prevent an independent test from only one server. This needs at least two servers, e.g., the front server and the back server, and these servers are not allowed to collude. The general two-server construction against insider attacks is illustrated in Fig. 6. Under this framework, two-server PEKS schemes (Wang and Tu 2014; Chen et al. 2015) were proposed by borrowing the property of the secret sharing and using smooth projective hash function, respectively.

- 1: $\text{Setup}(\lambda) \rightarrow SP$
- 2: $\text{KeyGen}_{\text{Sender}}(SP) \rightarrow (pk_E, sk_E)$
- 3: $\text{KeyGen}_{\text{Receiver}}(SP) \rightarrow (pk_R, sk_R)$
- 4: $\text{PEKS}(SP, pk_R, pk_E, w) \rightarrow CT$
- 5: $\text{Trapdoor}(sk_R, pk_E, w) \rightarrow T$
- 6: $\text{Test}(SP, CT, T, pk_R, pk_E) \rightarrow 1/0$, where pk_R, pk_E are optional.

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 5 General authentication-based construction against insider attacks

```

1:  $\text{Setup}(\lambda) \rightarrow SP$ 
2:  $\text{KeyGen}(SP) \rightarrow (pk_{FS}, sk_{FS}), (pk_{BS}, sk_{BS})$ 
3:  $\text{PEKS}(SP, pk_{FS}, pk_{BS}, w) \rightarrow CT$ 
4:  $\text{Trapdoor}(SP, pk_{FS}, pk_{BS}, w) \rightarrow T$ 
5:  $\text{FTest}(SP, CT, T, sk_{FS}) \rightarrow C_{ITS}$ 
6:  $\text{BTest}(SP, sk_{BS}, C_{ITS}) \rightarrow 1/0$ 

```

Keyword Attacks and Privacy Preserving in Public-Key-Based Searchable Encryption, Fig. 6 General two-server construction against insider attacks

Future Directions

Based on the above investigations of current PEKS and its variants, we list several future directions to advance the development of PEKS.

- KGA over the multi-keyword setting. The originally defined KGA is launched on a keyword. However, for the multi-keyword PEKS variants, such as PECKS, guessing multiple keywords via repeated attacks seems to be less efficient. The first problem is to explore whether it is possible to launch KGA once and guess all of the keywords. If yes, how to resist KGA in multi-keyword setting? The previous solutions will not work because they aim at only one keyword. Conducting research on such a KGA variant will make a significant contribution to public-key-based searchable encryption.
- Improving search efficiency. The time complexity of the search in current schemes is linear in the number of data, even if they can achieve KGA resilience. However, in the big data era, large-scale data need to be stored on the server, and the one-by-one test is impractical. How to reduce the complexity in searching large-scale data is an open problem in the future.
- Search over encrypted media. The current searchable encryption schemes are based on the keyword. This is more suitable for file-based systems. However, some data is presented with image or video. Search over encrypted media information and the privacy preserving become challenging since these media contents have different formats from

the keyword and cannot be guessed from a low-entropy space. Research on media-based will widen the scope of searchable encryption.

Conclusion

In this survey, we focused on the keyword privacy issues of public-key encryption with keyword search. We analyzed the keyword-guessing attacks as well as corresponding reasons. We summarized enhancement achievements in terms of different types of attackers. Finally, we discussed several future directions for a better development of searchable encryption.

References

- Bethencourt J, Song DX, Waters B (2006) New constructions and practical applications for private stream searching (extended abstract). In: S&P 2006, pp 132–139
- Boneh D, Franklin MK (2001) Identity-based encryption from the weil pairing. In: Kilian J (ed) CRYPTO. LNCS 2001, vol 2139, pp 213–229
- Boneh D, Waters B (2007) Conjunctive, subset, and range queries on encrypted data. In: Vadhan SP (ed) TCC 2007. LNCS, vol 4392, pp 535–554
- Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: Cachin C, Camenisch J (eds) EUROCRYPT 2004. LNCS, vol 3027, pp 506–522
- Bost R, Minaud B, Ohremenko O (2017) Forward and backward private searchable encryption from constrained cryptographic primitives. In: Thuraisingham BM, Evans D, Malkin T, Xu D (eds) CCS 2017, pp 1465–1482
- Byun JW, Rhee HS, Park H, Lee DH (2008) Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker W, Petkovic M (eds) Third VLDB workshop, SDM 2006. LNCS, vol 4165, pp 75–83
- Chen R, Mu Y, Yang G, Guo F, Wang X (2015) A new general framework for secure public key encryption with keyword search. In: Foo E, Stebila D (eds) ACISP 2015. LNCS, vol 9144, pp 59–76
- Fang L, Susilo W, Ge C, Wang J (2013) Public key encryption with keyword search secure against keyword guessing attacks without random oracle. Inf Sci 238:221–241
- Huang Q, Li H (2017) An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. Inf Sci 403:1–14

- Jiang P, Mu Y, Guo F, Wang X, Wen Q (2016a) Online/offline ciphertext retrieval on resource constrained devices. *Comput J* 59(7):955–969
- Jiang P, Mu Y, Guo F, Wen Q (2016b) Public key encryption with authorized keyword search. In: Liu JK, Steinfeld R (eds) ACISP 2016. LNCS, vol 9723, pp 170–186
- Jiang P, Mu Y, Guo F, Wen Q (2017a) Private keyword-search for database systems against insider attacks. *J Comput Sci Technol* 32(3):599–617
- Jiang P, Mu Y, Guo F, Wen Q (2017b) Secure-channel free keyword search with authorization in manager-centric databases. *Comput Secur* 69:50–64
- Kamara S, Moataz T (2017) Boolean searchable symmetric encryption with worst-case sub-linear complexity. In: Coron J, Nielsen JB (eds) EUROCRYPT 2017. LNCS, vol 10212, pp 94–124
- Kim KS, Kim M, Lee D, Park JH, Kim W (2017) Forward secure dynamic searchable symmetric encryption with efficient updates. In: Thuraisingham BM, Evans D, Malkin T, Xu D (eds) CCS 2017, pp 1449–1463
- Park DJ, Kim K, Lee PJ (2004) Public key encryption with conjunctive field keyword search. In: Lim CH, Yung M (eds) WISA 2004. LNCS, vol 3325, pp 73–86
- Rhee HS, Park JH, Susilo W, Lee DH (2009a) Improved searchable public key encryption with designated tester. In: Li W, Susilo W, Tupakula UK, Safavi-Naini R, Varadharajan V (eds) ASIACCS 2009, pp 376–379
- Rhee HS, Susilo W, Kim H (2009b) Secure searchable public key encryption scheme against keyword guessing attacks. *IEICE Electron Expr* 6(5):237–243
- Rhee HS, Park JH, Susilo W, Lee DH (2010) Trapdoor security in a searchable public-key encryption scheme with a designated tester. *J Syst Softw* 83(5):763–771
- Shi J, Lai J, Li Y, Deng RH, Weng J (2014) Authorized keyword search on encrypted data. In: Kutylowski M, Vaidya J (eds) ESORICS 2014. LNCS, vol 8712, pp 419–435
- Song DX, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: S&P 2000, pp 44–55
- Sun W, Yu S, Lou W, Hou YT, Li H (2016) Protecting your right: verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. *IEEE Trans Parallel Distrib Syst* 27(4): 1187–1198
- Sun L, Xu C, Zhang M, Chen K, Li H (2018) Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation. *Sci China Inf Sci* 61(3):038106
- Wang CH, Tu TY (2014) Keyword search encryption scheme resistant against keyword-guessing attack by the untrusted server. *J Shanghai Jiaotong Univ (Sci)* 19(4):440–442
- Yang Y, Ma M (2016) Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds. *IEEE Trans Inf Forensics Secur* 11(4):746–759
- Yau W, Heng S, Goi B (2008) Off-Line keyword guessing attacks on recent public key encryption with keyword

search schemes. In: Rong C, Jaatun MG, Sandnes FE, Yang LT, Ma J (eds) ATC 2008. LNCS, vol 5060, pp 100–105

Zheng Q, Xu S, Ateniese G (2014) VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In: INFOCOM 2014, pp 522–530

K Nearest Neighbor Query

► Query Processing – *k*NN

kNN Query

► Query Processing – *k*NN

K

Knowledge Base

► Reasoning at Scale

Knowledge Base Embeddings

► Knowledge Graph Embeddings

Knowledge Discovery

► Big Data and Recommendation

Knowledge Graph Embeddings

Paolo Rosso, Dingqi Yang, and Philippe Cudré-Mauroux
eXascale Infolab, University of Fribourg,
Fribourg, Switzerland

Synonyms

Knowledge base embeddings; RDF graph embeddings

Definitions

Knowledge graph embeddings: a vector representation of entities and relations in a knowledge graph that preserves the inherent structure of the knowledge graph as well as the reasoning ability over the graph.

Overview

With the growing popularity of multi-relational data on the Web, knowledge graphs (KGs) have become a key data source in various application domains, such as Web search, question answering, and natural language understanding. In a typical KG such as Freebase (Bollacker et al. 2008) or Google’s Knowledge Graph (Google 2014), entities are connected via relations. For example, *Bern* is capital of *Switzerland*. Formally, a popular approach to represent such relational data is to use the Resource Description Framework. It defines a fact as a triple (subject, predicate, and object), which is also known as head, relation, and tail or (h, r, t) for short. Following the above example, the head, relation, and tail are *Bern*, *capitalOf*, and *Switzerland*, respectively. With a considerable number of entities and relations (e.g., Google’s Knowledge Graph has more than 18 billion of triples with 570 million of entities and 35,000 of relations by the end of 2014), KGs now become a valuable information source that can empower many semantic Web applications.

Despite the importance of building large-scale KGs, their symbolic and logical frameworks are not flexible enough to be compatible with modern statistical and machine learning techniques which require often numerical inputs. In this context, knowledge graph embeddings that project entities and relations in a KG into a low-dimensional continuous vector space have attracted much attention. One of the key benefits of such numeric representation is that they can easily serve as input to classical statistical and machine-learning approaches. The learnt entity and relation embeddings can thus be used in different tasks, such as KG completion (Bordes et al. 2013),

relation extraction (Weston et al. 2013), entity classification, and entity resolution (Nickel et al. 2011).

In the following, we first discuss typical KG embedding models and then their extensions by integrating additional data sources. We then summarize the applications of KG embeddings.

Learning Knowledge Graph Embeddings

Learning KG embeddings consists in two key steps in general:

1. Defining a KG embedding model with a specific scoring function, which computes the probability that a given triple is true;
2. Initializing entity and relation embeddings (e.g., vectors) according to the KG embedding model and learning those embeddings by maximizing the sum of the scoring function over all triples in the KG. Triples appearing in the KG will have higher scores than the triples that do not exist in the KG.

KG Embedding Models

Depending on the type of scoring function, there are two categories of embedding models: translational distance models and semantic matching models. Translational distance models, such as *TransE* (Bordes et al. 2013), use a scoring function that measures the distance between two entities, while semantic matching models, such as *RESCAL*, use a scoring function that measures the similarity of the facts.

Translational Distance Model: TransE and Its Extensions

TransE is a representative translational distance model that projects entities and relations onto a unique vector space. In this model, the head h and the tail t of a triplet are connected by their relation r , holding the fact that the embedding of t should be similar to the embedding of h plus the embedding of r (i.e., $h + r \approx t$). The proposed idea is based on the vector-offset method for identifying linguistic regularities in continuous

space word representations (Mikolov et al. 2013), for example, *USA – dollar ≈ Japan – yen*. In a KG, this analogy holds since through the *currencyOf* relation, we get *dollar + currencyOf ≈ USA* and *yen + currencyOf ≈ Japan*. In this way, the scoring function is defined as the negative distance between the sum of the head and the relation, subtracted by the tail:

$$f_r(h, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1/2} \quad (1)$$

Following this initial idea, several techniques improved TransE by designing sophisticated scoring functions that are able to capture complex KG structures, in particular multi-mapping relations (one-to-many, many-to-one, or many-to-many). TransH (Wang et al. 2014b), for instance, suggests a new approach by projecting the relations on different hyperplanes in order to capture many-to-many mapping properties of some relations; TransR (Lin et al. 2015b) defines a mapping matrix and a vector for every relation; TransD (Ji et al. 2015) introduces dynamic matrices for each entity-relation pair by considering the diversity of entities and relations simultaneously. Sophisticated scoring functions can indeed improve the KG embeddings in some downstream learning tasks, though they also increase the complexity of the embedding models.

Semantic Matching Model: RESCAL and Its Extensions

RESCAL is a tensor factorization model for KG embeddings, which decomposes a three-way tensor consisting of head, relation, and tail dimensions. RESCAL generates for each entity a vector and for each relation a matrix capturing the interaction between the entities. The proposed model allows for discovering the correlation between multiple interconnected entities. The model represents facts via a tensor product with a corresponding scoring function defined as follows:

$$f_r(h, t) = \mathbf{h}^\top \mathbf{M}_r \mathbf{t} = \sum_{i=1}^d \sum_{j=1}^d [\mathbf{M}_r]_{ij} \cdot [\mathbf{h}]_i \cdot [\mathbf{t}]_j \quad (2)$$

where h and t are the vectors of the head and tail, respectively, and M_r is the matrix that represents the relations. d refers to the dimension of the embeddings.

Several pieces of work extend RESCAL by designing customized tensor factorization models. DistMult (Yang et al. 2014) simplifies the RESCAL model by using a bilinear formulation, showing similar performance with less parameters (more efficient in the learning process). However, the model works with symmetric relations only. Trouillon et al. (2016) propose a model called Complex Embeddings (ComplEx) that extends DistMult in order to model asymmetric relations. ComplEx introduces a complex space in which head, relation, and tail embeddings are represented. In this model, the scoring function generates different scores from facts with asymmetric relations. Neural tensor network (NTN) (Socher et al. 2013) is another model with a neural network architecture. For each fact, the embedding vectors of its head and tail are fed into the input layer of a neural network and then mapped onto its hidden layer combined with a relation-specific tensor. Finally, the output layer generates a score for each fact.

Learning Process

The goal of the learning process is to maximize the sum of the scoring function over all triples in the KG. Typical examples of optimization algorithms used in this context include stochastic gradient descent (SGD) (Robbins and Monroe 1951), Broyden-Fletcher-Goldfarb-Shanno (BFGS) (Battiti and Masulli 1990), and AdaGrad (Duchi et al. 2011). In order to accelerate the training process, negative sampling techniques can be applied by replacing the head, the relation, or the tail of a given fact. These generated triples are called negative samples. There are two main methods of generating negative samples, based on the *open-world assumption (OWA)* and on the *closed-world assumption (CWA)*.

Open-World Assumption

The open-world assumption assumes that the KG only contains true facts and the facts that do not appear can either be false or just missing

(Drumond et al. 2012). Under this assumption, a negative fact is probabilistically generated given a positive fact by randomly corrupting its head, relation, or tail. The entity and relation representations are learned by minimizing a loss function defined based on the scoring function, such as the logistic loss defined as follows:

$$\min_{h,r,t} \sum_{(h,r,t) \in D^+ \cup D^-} \log(1 + \exp(-y_{hrt} \cdot f_r(h, t))) \quad (3)$$

where D^+ and D^- are positive and negative training samples and y_{hrt} is equal to 1 if the label is positive, -1 otherwise. The logistic loss can be optimized using, for instance, stochastic gradient descent in mini-batch mode. In the training phase, a set of true facts are sampled and a set of negative facts get generated, and the embeddings can be iteratively updated with a fixed or adaptive learning rate.

Closed-World Assumption

The closed-world assumption assumes that all the facts that are not in the KG are negative samples (i.e., assuming the KG is complete). The entity and relation representations are learned by minimizing, for example, the squared loss:

$$\min_{h,r,t} \sum_{h,t \in \mathbb{E}, r \in \mathbb{R}} (y_{hrt} - f_r(h, t))^2 \quad (4)$$

where \mathbb{E} and \mathbb{R} are the set of entities and relations, respectively. y_{hrt} is equal to 1 if the triplet appears in the KG, 0 otherwise. In addition, Nickel and Tresp (2013) propose the logistic loss and (Miettinen 2011) the absolute loss as alternatives to the squared loss.

In summary, the closed-world assumption usually has more limitations than the open-world assumption, as it penalizes the missing true facts from a KG. In practice, despite their tremendous size, modern KGs all suffer from incompleteness issues (West et al. 2014). Consequently, the open-world assumption is more realistic for most settings and thus performs better on average than the closed-world assumption (Guo et al. 2017).

KG Embedding Extension by Integrating Additional Information

When learning KG embeddings, additional information, such as entity types, relation paths, textual descriptions, or logical rules, can be added to the embedding model to improve the quality of the embeddings on certain tasks.

Entity types represent semantic categories the entity belongs to. For example, the entity *PresidentObama* can be annotated as a *PERSON* entity type. This piece of information can be incorporated in different ways. Nickel et al. (2012) use the entity type as a relation and its corresponding facts (h, r, t) as training example. Guo et al. (2015) propose a method in which entities of the same type are close to each other in the vector space. The entity type can also be used to set constraints for different relations. For example, Xie et al. (2016) use this constraint to generate correct negative samples by filtering out triples with incorrect entity types.

Relation paths refer to a sequence of relations between two entities. The multi-hop relationships contain useful information that can be used for KG completion. For example, Lao and Cohen (2010) predict the relation between entities using a path ranking algorithm that connects two entities. More precisely, the relations can be represented as vectors or matrices, and their addition or multiplication can be used to compose a path as vector or matrix composition. Lin et al. (2015a) show a method to approximate the relation path via sampling and pruning. Along similar lines, Toutanova et al. (2016) propose an algorithm that incorporates paths with specific lengths and intermediate entities in the model.

Most of the KGs contain entity descriptions that can be used to enrich the semantic information of the entities. External information sources, such as Wikipedia articles (Wang et al. 2014a) or news releases (Socher et al. 2013), can be used to extend the entity description by providing richer textual information. A representative work by Wang et al. (2014a) introduces a model that combines text corpus and KG to align them in the same vector space and creates KG embeddings and text embeddings. The model includes

three main parts: a KG model, a text model, and an alignment model. Specifically, the KG model is used to generate embeddings of entities and relations in the KG, while the text model is used to generate embeddings from the text corpus. Finally, the alignment model is used to align the KG embedding and text embedding in the same vector space using different alignment mechanisms, such as entity name and Wikipedia anchors. In this way, the model is able to predict out-of-KG entities (phrases not stored in the KG but that appear in the text).

Logical rules are another type of information that could be integrated into embedding models. For instance, if two entities are connected by the relation *HasWife*, then they should also be connected by the relation *HasSpouse*. There exist systems, such as *WARMR* (Dehaspe and Toivonen 1999), *ALEPH* (Muggleton 1995), and *AMIE* (Galárraga et al. 2013), that can automatically extract such kinds of relations. Richardson and Domingos (2006) prove that the logical rules contain rich information and that they can be used to acquire and infer further knowledge. Following this idea, Wang et al. (2015) propose an approach seamlessly incorporating logic rules into KG embedding models by reducing the solution space and thus improving the inference accuracy for knowledge base completion tasks. Guo et al. (2016) propose a model that embeds KG and logical rules in a unified framework. Specifically, logical rules are first instantiated into ground rules, for example, $\text{HasWife}(x,y)$ implies $\text{HasSpouse}(x,y)$ and vector embeddings are introduced for entity pairs. For each fact of the logical rule, a score is computed in order to indicate whether the ground rule is satisfied or not. The embedding model is then learned based on the unified facts and rules. In this way, the model is more effective for knowledge acquisition and inference as the embeddings are compatible with both facts and rules.

In addition to the information described above, further types of information that can be added to the embedding models include entity attributes, temporal information, and graph structures. Nickel et al. (2012) highlight the fact that entity attributes and relations must be separated. The

authors propose a new algorithm to handle attributes efficiently. Jiang et al. (2016) show that the KG facts are often time-sensitive and that they develop a time-aware knowledge base embedding approach by taking advantage of the time at which facts have occurred. The proposed solution forces the embeddings to be temporally consistent by using temporal constraints to model the relations. Feng et al. (2016) show a graph-aware approach that learns entity and relation embeddings by leveraging the relation paths and edge context (i.e., all the relations that connect an entity). The intuition behind this approach is that all the relations linking to and from an entity are representative of that entity.

KG Embedding Applications

Typical applications of KG embedding include link prediction, triple classification, entity classification, and entity resolution.

- Link prediction (also called KG completion) attempts to discover missing facts based on the contents of the KG. Specifically, it predicts an entity given a relation and a second entity, i.e., given (r, t) , it predicts h , denoted also as $(?, r, t)$, or given (h, r) , it predicts t , denoted also as $(h, r, ?)$. Lin et al. (2015a) define this task as entity prediction, while Bordes et al. (2014b) define it as entity ranking. A similar approach predicts a relation given its head and tail entities, denoted also as $(h, ?, t)$, which is similar also to relation prediction (Xie et al. 2016). In order to evaluate the results generated by this task, a common practice is to store in a list all the answers and see the rank of the correct answer. Several evaluation metrics can be used in this context, such as *Hits@n* that considers only the ranks smaller than n , or the *mean rank*, that is, the average of the predicted ranks.
- Triple classification aims at determining whether a triple appears in a KG. More precisely, triple classification can be performed based on the score of a candidate triple

(h, r, t) that can be easily computed using the scoring function. In this way, an unseen fact can be either true if its score is higher than a threshold and false otherwise. Traditional evaluation metrics can be used in this task, for example, mean average precision (Guo et al. 2016) or micro- and macro-averaged accuracy (Guo et al. 2015).

- Entity classification classifies entities into different semantic categories. Concretely, the type of an entity is usually denoted using a *IsA* relation, and entity classification can thus be seen as a particular case of the link prediction task, in which only $(h, \text{IsA}, ?)$ triples are predicted.
- Entity resolution verifies whether two entities are actually referring to the same object or not. Bordes et al. (2014b) tackle this problem by considering a scenario in which the relation of two equivalent entities is explicitly denoted as *EqualTo*. By learning the embedding for this type of relation, the problem of entity resolution becomes a triple classification problem. Fundamentally, the triple classification problem judges whether the fact $(h, \text{EqualTo}, t)$ is true or not. Alternatively, Nickel et al. (2011) propose a different approach that computes the similarity between two entities and use the score to calculate the likelihood that two entities refer to the same object. This method works even if the relation *EqualTo* is not encoded in the KG.

KG embeddings can also be applied to other application domains beyond KGs. Three most popular out-of-KG applications are relation extraction, question answering, and recommendation systems.

- Relation extraction tries to discover relations from text where entities have already been identified. Weston et al. (2013) propose a method to extract relations by combining TransE and text, showing that the integration of TransE and a traditional

text-based extractor can actually improve the performance of relation extraction.

- Question answering refers to the task of answering questions over KGs. Given a question in plain text, a fact or a set of facts containing the correct answer is extracted as an answer. This task is challenging because of the extended variability of natural language text used to formulate the question and of the extensive size of the KGs. A successful solution that involves KG embedding is proposed by Bordes et al. (2014a), which learns embeddings in order to put questions and corresponding answers closer in the vector space. Given a question and an answer, the model generates a high score if the answer is correct, low score otherwise. The results show that, by involving the KG, the task is successfully performed without using any rules or additional tagging step as most traditional question answering applications do.
- Recommendation systems suggest users a list of items according to the users' preferences. Collaborative filtering techniques are often used to perform recommendations based on the historical interaction between users and items. However, user-item interactions are often sparse, leading to unsatisfactory performance. To alleviate this issue, hybrid recommendation systems were developed by adding auxiliary information (Yu et al. 2014). Zhang et al. (2016) propose a hybrid recommendation system that integrates a KG. More precisely, the hybrid recommendation system models structural knowledge by applying a KG embedding technique such as *TransR* in order to learn the representation of each item. Similarly, the users are represented by vectors, and each item is represented by its KG vector representation plus an offset. Finally, the preference of a user for a specific item is computed as a product of the user and item vectors. In this way, the hybrid recommendation system automatically extracts semantic representations from facts in the KG to improve the quality of the recommendation system.

Conclusion

With the booming of multi-relational data on the Web, knowledge graphs have become an important data source empowering many applications. However, the symbolic and logical representation of KGs makes it difficult to take them as input to machine-learning or processing pipelines. To tackle this issue, knowledge graph embedding techniques were proposed to project entities and relations from a KG into a low-dimensional continuous vector space while still preserving the inherent structure of the KG and reasoning capabilities over the KG. The learnt embeddings have been successfully used in both KG-reasoning applications, such as link prediction, triple classification, entity classification, and entity resolution, and out-of-KG applications, such as relation extraction, question answering, and recommendation systems.

Cross-References

- [Automated Reasoning](#)
- [Big Data and Recommendation](#)

References

- Battiti R, Masulli F (1990) BFGS optimization for faster and automated supervised learning. In: International neural network conference. Springer, pp 757–760
- Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, pp 1247–1250
- Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O (2013) Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems, pp 2787–2795
- Bordes A, Chopra S, Weston J (2014a) Question answering with subgraph embeddings. arXiv preprint, arXiv:14063676
- Bordes A, Glorot X, Weston J, Bengio Y (2014b) A semantic matching energy function for learning with multi-relational data. Mach Learn 94(2):233–259
- Dehaspe L, Toivonen H (1999) Discovery of frequent datalog patterns. Data Min Knowl Disc 3(1):7–36
- Drumond L, Rendle S, Schmidt-Thieme L (2012) Predicting RDF triples in incomplete knowledge bases with tensor factorization. In: Proceedings of the 27th annual ACM symposium on applied computing. ACM, pp 326–331
- Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
- Feng J, Huang M, Yang Y et al (2016) Gake: graph aware knowledge embedding. In: Proceedings of COLING 2016, the 26th international conference on computational linguistics: technical papers, pp 641–651
- Galárraga LA, Teflioudi C, Hose K, Suchanek F (2013) AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In: Proceedings of the 22nd international conference on World Wide Web. ACM, pp 413–422
- Google (2014) https://www.google.com/intl/bn/inside_search/features/search/knowledge.html
- Guo S, Wang Q, Wang B, Wang L, Guo L (2015) Semantically smooth knowledge graph embedding. In: ACL, vol 1, pp 84–94
- Guo S, Wang Q, Wang L, Wang B, Guo L (2016) Jointly embedding knowledge graphs and logical rules. In: EMNLP, pp 192–202
- Guo S, Wang Q, Wang B, Wang L, Guo L (2017) SSE: semantically smooth embedding for knowledge graphs. IEEE Trans Knowl Data Eng 29(4):884–897
- Ji G, He S, Xu L, Liu K, Zhao J (2015) Knowledge graph embedding via dynamic mapping matrix. In: ACL, vol 1, pp 687–696
- Jiang T, Liu T, Ge T, Sha L, Li S, Chang B, Sui Z (2016) Encoding temporal information for time-aware link prediction. In: EMNLP, pp 2350–2354
- Lao N, Cohen WW (2010) Relational retrieval using a combination of path-constrained random walks. Mach Learn 81(1):53–67
- Lin Y, Liu Z, Luan H, Sun M, Rao S, Liu S (2015a) Modeling relation paths for representation learning of knowledge bases. arXiv preprint, arXiv:150600379
- Lin Y, Liu Z, Sun M, Liu Y, Zhu X (2015b) Learning entity and relation embeddings for knowledge graph completion. In: AAAI, pp 2181–2187
- Miettinen P (2011) Boolean tensor factorizations. In: 2011 IEEE 11th international conference on Data mining (ICDM). IEEE, pp 447–456
- Mikolov T, Yih Wt, Zweig G (2013) Linguistic regularities in continuous space word representations. In: HLT-NAACL, vol 13, pp 746–751
- Muggleton S (1995) Inverse entailment and prolog. N Gener Comput 13(3):245–286
- Nickel M, Tresp V (2013) Logistic tensor factorization for multi-relational data. arXiv preprint, arXiv:13062084
- Nickel M, Tresp V, Kriegel HP (2011) A three-way model for collective learning on multi-relational data. In: ICML, vol 11, pp 809–816

- Nickel M, Tresp V, Kriegel HP (2012) Factorizing yago: scalable machine learning for linked data. In: Proceedings of the 21st international conference on World Wide Web. ACM, pp 271–280
- Richardson M, Domingos P (2006) Markov logic networks. *Mach Learn* 62(1):107–136
- Robbins H, Monro S (1951) A stochastic approximation method. *Ann Math Stat* 22:400–407
- Socher R, Chen D, Manning CD, Ng A (2013) Reasoning with neural tensor networks for knowledge base completion. In: Advances in neural information processing systems, pp 926–934
- Toutanova K, Lin V, Yih Wt, Poon H, Quirk C (2016) Compositional learning of embeddings for relation paths in knowledge base and text. In: ACL, vol 1
- Trouillon T, Welbl J, Riedel S, Gaussier É, Bouchard G (2016) Complex embeddings for simple link prediction. In: International conference on machine learning, pp 2071–2080
- Wang Z, Zhang J, Feng J, Chen Z (2014a) Knowledge graph and text jointly embedding. In: EMNLP, vol 14, pp 1591–1601
- Wang Z, Zhang J, Feng J, Chen Z (2014b) Knowledge graph embedding by translating on hyperplanes. In: AAAI, pp 1112–1119
- Wang Q, Wang B, Guo L (2015) Knowledge base completion using embeddings and rules. In: IJCAI, pp 1859–1866
- West R, Gabrilovich E, Murphy K, Sun S, Gupta R, Lin D (2014) Knowledge base completion via search-based question answering. In: Proceedings of the 23rd international conference on World Wide Web. ACM, pp 515–526
- Weston J, Bordes A, Yakhnenko O, Usunier N (2013) Connecting language and knowledge bases with embedding models for relation extraction. arXiv preprint, arXiv:13077973
- Xie R, Liu Z, Sun M (2016) Representation learning of knowledge graphs with hierarchical types. In: IJCAI, pp 2965–2971
- Yang B, Yih Wt, He X, Gao J, Deng L (2014) Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint, arXiv: 14126575
- Yu X, Ren X, Sun Y, Gu Q, Sturt B, Khandelwal U, Norick B, Han J (2014) Personalized entity recommendation: a heterogeneous information network approach. In: Proceedings of the 7th ACM international conference on web search and data mining. ACM, pp 283–292
- Zhang F, Yuan NJ, Lian D, Xie X, Ma WY (2016) Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 353–362

Knowledge Graphs in the Libraries and Digital Humanities Domain

Bernhard Haslhofer¹, Antoine Isaac², and Rainer Simon¹

¹Austrian Institute of Technology, Vienna, Austria

²Europeana Foundation, The Hague, The Netherlands

Definitions

Knowledge graphs represent concepts (e.g., people, places, events) and their semantic relationships. As a data structure, they underpin a digital information system, support users in resource discovery and retrieval, and are useful for navigation and visualization purposes. Within the libraries and humanities domain, knowledge graphs are typically rooted in *knowledge organization systems*, which have a century-old tradition and have undergone their digital transformation with the advent of the Web and Linked Data. Being exposed to the Web, metadata and concept definitions are now forming an interconnected and decentralized global knowledge network that can be curated and enriched by community-driven editorial processes. In the future, knowledge graphs could be vehicles for formalizing and connecting findings and insights derived from the analysis of possibly large-scale corpora in the libraries and digital humanities domain.

Overview

The term “knowledge graph” was popularized in 2012 (Singhal 2012) with the announcement of the “Google Knowledge Graph,” which gathers and formalizes information from several sources in order to enhance search results. Knowledge graphs can be generic and cover a broad range of domains, or they can be tailored to a specific

domain or application context. They typically integrate data from multiple, heterogeneous sources and provide both a human-interpretable representation and a formalized machine-readable basis for information retrieval tasks, such as (latent) semantic indexing, classification, or query recommendation. Well-known examples for openly available, generic knowledge graphs are DBpedia (Bizer et al. 2009a), Freebase (Bollacker et al. 2008), or Wikidata (Vrandečić and Krötzsch 2014).

Within the libraries and digital humanities domain, the concept of a knowledge graph is deeply rooted in *knowledge organization systems*, which is an umbrella term for vocabularies such as a classification schemes, thesaurus, or glossary.

Knowledge organization systems have a centuries-old tradition in the library domain, where they have been used in metadata descriptions to organize resources and facilitate their discovery and retrieval. With the advent of Linked Data (Bizer et al. 2009b), knowledge organization systems – and metadata descriptions making use of them – have undergone a digital transformation and entered the realm of the World Wide Web. By linking them with semantically related Web resources within and outside the library domain, they are now forming an interconnected and decentralized global knowledge graph.

The development of knowledge graphs also has a long-standing tradition in the humanities. For a long time, scholarly efforts have been concerned with the curation of authoritative data on, e.g., places or people within a specific domain, resulting in taxonomies, gazetteers, and prosopographies. In terms of their digital transformation, these knowledge graphs have been following a similar pattern as knowledge organization schemes in libraries: they are increasingly being published according to Linked Data principles and connected to other knowledge graphs on the Web, using shared semantic concepts.

Members of both domains soon realized that knowledge graphs can be curated and enriched

by community-driven editorial processes, similar to those of Wikipedia. As a result, processes emerged that either allow users to directly edit a knowledge graph or provide mechanisms such as *semantic tagging* to support enriching corpora with concepts from knowledge graphs.

Cross-domain knowledge graphs, as they are emerging in the libraries and digital humanities domain, open a whole new spectrum of research opportunities. They can, for instance, inform the design of quantitative analytics tasks being applied on possibly large-scale document corpora. Taking into account the numerous digitization efforts in the library and humanities domain, knowledge graphs can be vehicles for connecting and exchanging findings as well as factual knowledge gained by applying those task on corpora. This can also stimulate the development of novel, mixed-method research methodologies.

K

Knowledge Organization Systems in Libraries

Libraries have long identified the value of maintaining (meta)data about their holdings. Over centuries the classical form of the book-based catalog has evolved into card-based catalogs and then electronic catalogs organized around the *record* unit, representing data about one of the library's holdings (a monograph, a journal issue, etc.). Library science developed resources and methods to rationalize the way these data were produced and exploited. The first of them were designed for classification and subject indexing, such as the Dewey Decimal Classification (DDC), the Universal Decimal Classification (UDC), or the Library of Congress Subject Headings (LCSH). Classification schemes, subject heading lists, thesauri, and other taxonomies were engineered to guide the description of the subjects of library assets (or their type). Name authority files and gazetteers played a similar role as organized listings of names for persons, organizations, and places. All these are composed

of elements (classes, concepts, names, etc.) that are provided with lexical information (variants, synonyms) or semantic information (especially, hierarchical or associative links), which qualifies them as knowledge organization systems. They also constitute vast bodies of knowledge, with dozens of thousands of concepts (LCSH contained 342,107 authority records in April 2017) that are richly described (UDC classes exist in over 40 languages).

In parallel, the structure of records was also the subject of much rationalization efforts, with description rules being agreed at the level of thematic or geographic communities. As libraries became (early) adopters of electronic information systems, these gave rise to formats geared for exchange of bibliographic metadata, such as MARC, which defines lists of codes for the fields in records (title, author, subject, etc.).

The need for structure and control within library systems came at the same time as a need to share and collaborate. Rather than each developing their own knowledge organization systems, libraries sought to share and reuse existing ones: classifications like UDC, thesauri like AGROVOC, and subject heading lists like LCSH or RAMEAU are used in entire networks of libraries. They can be seen as collaboratively built systems, as all the organizations that use them can contribute updates, even as one of them plays a leading role for maintenance and quality control. A similar form of sharing happens at the level of the datasets describing assets: *union cataloging* systems have been put in place so that consortia of libraries can refer to an existing description of a book, possibly after adding to it, rather than creating a new – probably poorer – one from scratch. This pooling of data gives even more importance to the knowledge organization systems backing these datasets and raised the motivation for enhancing their coverage and quality over time.

Of course, full standardization is not attainable, and overlapping vocabularies have been built, e.g., specific to a (sub-)domain or a (group of) country(ies). This became a problem when libraries needed a higher level of interoperability for their data. In some cases it has been ad-

dressed: the Virtual International Authority File (VIAF) has managed to consolidate links between the reference lists of persons and organization from dozens of (mostly national) libraries into a multilingual dataset (43 million clusters of links in 2014). MACS (Multilingual Access to Subjects) was another notable project, in which the national libraries of Switzerland, France, and Germany have teamed with the British Library to create thousands of links across the subject heading lists LCSH, RAMEAU, and SWD.

Considering the long-standing knowledge organization tradition in libraries, one can conclude that libraries have established systems resembling the characteristics of today's knowledge graphs. And indeed the Mundaneum, an early twentieth-century attempt to build a universal repository of facts, heavily inspired by library science, has been sometimes acknowledged as a precursor of the Semantic Web notion. Yet, as the W3C Incubator Group on Library Linked Data (W3C 2011) has put it, more than 20 years after the invention of the Web, the library and Semantic Web communities had similar metadata concepts but still different terminology; traditional library standards were designed only for the library community; library data was not well integrated with Web resources and still expressed primarily in presentation-oriented, natural-language text.

Library Linked Data

The adoption of the Linked Data principles is a major step in the transition from library-centric knowledge organization systems to domain-spanning, openly available, and easily accessible knowledge graphs. These principles postulate a conceptual representation of library objects and concepts as first-class Web resources and then propose (i) the assignment of unique Web identifiers (URI) to these resources, (ii) the provision of machine-readable metadata describing these resources, and (iii) linkage of resources with semantically related resources in other datasets or knowledge organization

systems. Those principles are being adopted in the publication of *metadata element sets* and *value vocabularies*, as detailed in the reports of the W3C Library Linked Data Incubator Group. This provided the basis for a number of data aggregation, linkage, and exposure efforts producing a large variety of *library linked datasets*.

Metadata Element Sets

A metadata element set provides elements (e.g., title, date, subject) to be used to describe a resource (e.g., a book), like the aforementioned MARC. Within the scope of Linked Data, metadata element sets are typically defined using RDF Schema (Brickley and Guha 2014) or the OWL Web Ontology Language (Hitzler et al. 2012). Besides supporting the definition of elements (properties), those schema or ontology definition languages also provide primitives for describing groups of related resources (classes) and the relationships between these resources. Important metadata element sets within the Library Linked Data field are the Dublin Core Metadata Element Set (DCMI 2012), the Bibliographic Framework Initiative (BIBFRAME) vocabulary, or the Resource Description and Access (RDA) element set. Interestingly, Dublin Core was developed at the time when the notions of the Semantic Web were articulated, with several key people involved in both initiatives, which positioned it as one of the most used vocabularies in Library Linked Data and beyond.

Value Vocabularies

A value vocabulary defines concepts that are used in the value space of bibliographic metadata records – cataloging rules in libraries often require terms of certain vocabularies to be used with certain metadata elements. Value vocabularies include knowledge organization systems such as introduced above, and their semantic expressiveness ranges from flat term definition lists (glossaries), over hierarchical structures such as classification schemes and taxonomies, to more connected vocabularies like thesauri.

Within the context of Linked Data, concepts defined as part of value vocabularies should be first-class resources identified by dereferenceable HTTP URIs. Concepts can be linked to other concepts within the same value vocabulary as well as to other concepts, possibly defined by another domain. The Simple Knowledge Organization System (SKOS) (Baker et al. 2013) has become the de facto standard for expressing the basic structure and content of controlled value vocabularies.

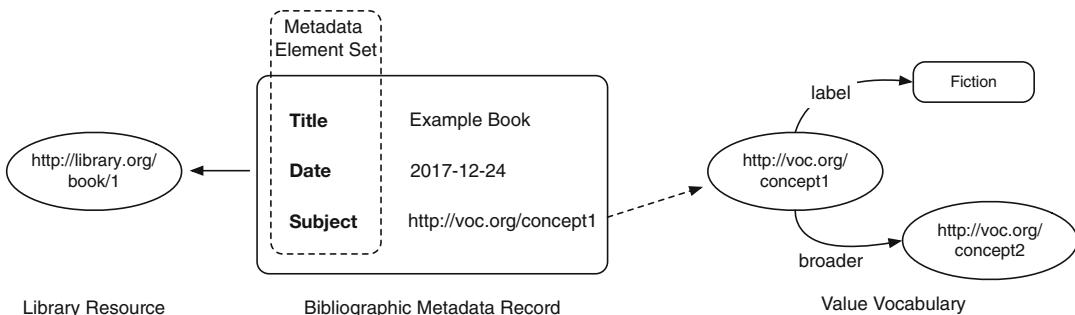
A number of value vocabularies established in the library field were made available as Linked Data: LCSH (Summers et al. 2008), the multilingual AGROVOC thesaurus (Caracciolo et al. 2013), VIAF, several national authority files (e.g., German GND), the Getty vocabularies (Getty 2017), as well as parts of the Dewey Decimal Classification (OCLC 2017).

The terms “ontology” and “vocabulary” are often used to refer both to metadata element sets and value vocabularies. Their role in the making of Linked Data differs quite significantly however. Descriptions typically reuse elements from standard metadata element sets in combination with elements from value vocabularies. Figure 1 shows an example of bibliographic metadata description with metadata fields taken from a metadata element set and a value vocabulary concept.

Library Linked Datasets

Within the last decade, a number of institutions have published collections of metadata records as Linked Data on the Web. Such records typically describe books, but other types of cultural objects can be described too.

The Hungarian and Swedish National (Malmsten 2008) libraries, who opened and exposed their OPAC catalog and corresponding authority files around the year 2008, were among the earlier adopters of the Linked Data principles. The British National Bibliography (BNB) (Deloit 2014) is a prominent Linked Library example which also links to external sources such as VIAF or LCSH. This was followed by many other data publication initiatives on an institutional or service level, including the Amsterdam



Knowledge Graphs in the Libraries and Digital Humanities Domain, Fig. 1 Library linked data example and notions

Museum (De Boer et al. 2012b), the CrossRef service, the Linked Open Library Data project, or Linked Art, which is a community working together to create a shared Linked Data model to describe art.

In parallel to institution- and service-level Linked Data publication initiatives, major metadata aggregation hubs across libraries and other institutions, such as Europeana (Haslhofer and Isaac 2011) and the Digital Public Library of America (DPLA) (DPLA 2017), started to expose collected cross-domain metadata records as Linked Open Data on a larger scale (as of 2017 Europeana and DPLA provide access to over 51 and 18 million objects, respectively) and to link them with other sources such as Wikidata. An important milestone achieved by these efforts was the wider adoption of public domain and (standard) open licenses (like the ones of Creative Commons) for the metadata. This has not only been a major business shift for libraries but also fulfilled two important preconditions for building large-scale, cross-domain knowledge graphs: the ability to mix and match library data with data from other sources and the possibility to edit and enhance knowledge graphs under control of a larger community.

Community-Driven Knowledge Graph Enrichment

In the early days of Linked Data, most knowledge graphs were published out of existing in-

stitutional repositories or extracted from public knowledge sources (e.g., Wikipedia) in a rather static, “push” fashion. The community soon realized however that knowledge graphs can be curated by community-driven editorial processes similar to those of Wikipedia.

A strong form of community-driven knowledge graph enrichment was provided by Freebase (Bollacker et al. 2008), which allowed users to enhance existing and add new facts to their knowledge graph. However, Freebase was acquired by Google and shut down its public API in 2016. An even stronger form is implemented by Wikidata (Vrandečić and Krötzsch 2014), in which the user community controls not only the data and schema but also the entire editorial process. Wikidata is also important as a case of interlinking vocabularies, tackling one of the issues in traditional library data. The Wikidata community-developed alignment tool Mix’n’Match, for instance, allows users to upload a vocabulary to Wikidata and create an alignment for it, which has helped to position Wikidata as a huge hub for Library Linked Data and beyond. Near-systematic references are now available between Wikidata entities and resources from vocabularies like VIAF and GND.

Within the digital libraries and digital humanities domains, community-driven knowledge graph enrichment mostly focused on involving users in linking resources from existing corpora with concepts in knowledge graphs. One possible enrichment technique is *semantic tagging*, which supports users in associating digital items or

fragments thereof with concepts expressed in existing knowledge graphs. That technique has been used for semantic tagging of historical maps (Haslhofer et al. 2013), images, and texts (Simon et al. 2015).

Knowledge Graphs in the Digital Humanities

The development and curation of domain-specific knowledge structures has traditionally been an important element of humanities scholarship. In many cases, these structures emerge as an implicit research output, e.g., when studying the interrelation between actors and events in a specific historical setting. In other cases, the development of a knowledge organization system as such – or components of it – is the main objective of the research endeavor. This is the case, for example, for the development of domain taxonomies, *gazetteers*, or *prosopographies* (dictionaries of people or groups of people). Initiatives such as the Tabula Imperii Romani (TIR-FOR 2016), the Tabula Imperii Byzantini (TIB 2017), the Prosopography of the Byzantine World (PBW 2011), the Great Britain Historical GIS (GBHGIS 2012), or the Treasury of Lives (ToL 2017) have been concerned exclusively – sometimes over a significant timespan, and long before the digital transformation – with the curation of authoritative data on places or people within their domain. Other efforts have focused on the translation of existing analog authority information to digital, some of them, such as the Pleiades Gazetteer of the Ancient World (Pleiades Contributors 2017) specifically as Linked Data.

The motivation behind the development of classification schemes and authority information in the humanities is much the same as in the library domain: a need for structure, control, and a common vocabulary to facilitate collaboration. Arguably, however, scholarly humanities research differs insofar as there is a much higher degree of specificity to particular – sometimes even niche – domains. The heterogeneous nature of research outputs, the interpretative quality of humanities research, and the fact that work is

often organized around the efforts of a single individual or small group, funded through time-limited grants, make a “global knowledge graph of the humanities” seem an unachievable goal. Yet the need to publish data under open licenses and build connections between datasets based on shared value vocabularies and metadata element sets is increasingly perceived in the community as key for enabling reuse, for the transparency of scholarly methods, and, ultimately, for sustainability of results. This trend is reflected in the rise of Linked Data as a theme at digital humanities events, conferences, and curricula, as much as in the emergence of community initiatives dedicated to establishing common interlinking standards and practices.

By and large, such initiatives advocate the idea of cross-domain linkage by means of shared name authority files, along with recommendations on metadata element use. Crucial to this effort are, on the one hand, generic knowledge graphs – DBpedia and Wikidata in particular. On the other hand, linked datasets from libraries or museums, such as VIAF, the Getty Thesaurus of Geographic Names, or the Getty Art and Architecture Thesaurus, have been gaining traction as an interconnecting “spine” through which community-specific datasets can build outbound links to contribute to a global graph.

As these networking efforts are enabling the aggregation of increasingly large corpora of cultural heritage content, the use of computational methods to study larger cultural phenomena is becoming more feasible (cf. Schich et al. 2014; Cook et al. 2012). New tools are transforming traditional scholarship, enabling scholars to identify and answer new research questions (cf. Siemens and Schreibman 2008; Bodenhamer et al. 2010; Bodard and Romanello 2016). Michel et al. (2010) popularized the term *culturomics* as “the application of high-throughput data collection and analysis to the study of human culture.” In this context, knowledge organization systems represent the crucial connecting medium within an ecology of independent initiatives that is gradually increasing mutual connectivity by creating and using Linked Open Data (Isaksen et al. 2014).

Future Directions of Research

Large-scale knowledge graphs, as they have been emerging in the libraries and digital humanities domain, and the publication of seminal papers (e.g., Michel et al. 2010) demonstrate how the application of quantitative analysis to large-scale corpora opens up a spectrum of possible new research questions that, up until now, were hard to answer with existing methods and primary sources. Data was manually curated and often bound to an institution or the scope of an individual researcher's project. Most studies in the humanities and related disciplines have focused on rather small corpora, while the constitution and maintenance of institutional knowledge organization systems and related datasets in libraries required years of work for a highly skilled and trained workforce.

Exploiting the opportunities of quantitative analysis methods poses a number of methodological, technical, and organizational challenges: first, novel serial analysis methods and tools are needed that support scholars in viewing, annotating, and systematically analyzing relevant parts of possibly large digitized corpora. Scholars could express relevance by selecting corresponding concept definitions in knowledge graphs. Second, scalable text-mining and machine-learning techniques are needed to systematically and efficiently analyze and compare the characteristics, contents, and relationships of concepts expressed in knowledge graphs within and across corpora. Third, algorithms are needed that support scholars in detecting, contextualizing, and analyzing various forms of expressions and associated narrative techniques in corpora spanning a long time period, in which the syntax and semantics may have been subject to constant change. Under this premise, future research could focus on:

- the development of tools and scalable techniques for aligning large-scale, possible multimedia corpora with concepts expressed in knowledge graphs
- the investigation of text mining algorithms that can learn from scholar's annotations and

support them in investigating semantic relationships extracted from large corpora

- the investigation of novel reconciliation mechanisms that ensure that institutional and community-curated knowledge graphs produced in different context are truly interoperable and do not lead to "competing" data offers
- the design of appropriate provenance tracking, crowd- or niche-sourcing (De Boer et al. 2012a) approaches, and validation mechanisms that ensure trust in data quality when data are curated by humans with different levels of expertise and/or result from automatic processes

The possible research directions described above show that there is a clear need for collaborative research among researchers from humanities, computer science, as well as library information science. This will also result in novel mixed qualitative and quantitative methods for the analysis of large-scale digitized corpora relevant to the humanities and related disciplines.

References

- Baker T, Bechhofer S, Isaac A, Miles A, Schreiber G, Summers E (2013) Key choices in the design of simple knowledge organization system (SKOS). *Web Semant Sci Serv Agents World Wide Web* 20(Suppl C):35–49
- Bizer C, Lehmann J, Kobilarov G, Auer S, Becker C, Cyganiak R, Hellmann S (2009a) DBpedia – a crystallization point for the Web of Data. *Web Semant Sci Serv Agents World Wide Web* 7(3):154–165. ISSN:1570-8268. <https://doi.org/10.1016/j.websem.2009.07.002>
- Bizer C, Heath T, Berners-Lee T (2009b) Linked data – the story so far. *Int J Semant Web Inf Syst* 5(3):1–22. <https://doi.org/10.4018/jswis.2009081901>
- Bodard G, Romanello M (2016) Digital classics outside the echo-chamber. Ubiquity Press, London. <http://www.oapen.org/record/608306>
- Bodenhamer DJ, Corrigan J, Harris TM (2010) The spatial humanities: GIS and the future of humanities scholarship. Indiana University Press, Bloomington
- Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, pp 1247–1250
- Brickley D, Guha RV (2014) RDF schema 1.1. W3C recommendation. <https://www.w3.org/TR/rdf-schema/>. Accessed 07 Dec 2017

- Caracciolo C, Stellato A, Morshed A, Johannsen G, Rajbhandari S, Jaques Y, Keizer J (2013) The AGROVOC linked dataset. *Semantic Web* 4(3):341–348
- Cook J, Das Sarma A, Fabrikant A, Tomkins A (2012) Your two weeks of fame and your grandmother's. In: Proceedings of the 21st international conference on World Wide Web (WWW'12). ACM, New York, pp 919–928. <https://doi.org/10.1145/2187836.2187959>
- DCMI (2012) Dublin core metadata element set. Online. <http://dublincore.org/documents/dces/>. Accessed 07 Dec 2017
- De Boer V, Hildebrand M, Arroyo L, De Leenheer P, Dijkshoorn C, Tesfa B, Schreiber G (2012a) Nichesourcing: harnessing the power of crowds of experts. In: ten Teije A, Völker J, Handschuh S, Stuckenschmidt H, d'Acquin M, Nikolov A, Aussena-Gilles N, Hernandez N (eds) Knowledge engineering and knowledge management, vol 7603. Springer, Berlin/Heidelberg, pp 16–20
- de Boer V. et al. (2012) Nichesourcing: Harnessing the Power of Crowds of Experts. In: ten Teije A. et al. (eds) Knowledge Engineering and Knowledge Management. EKAW 2012. Lecture Notes in Computer Science, vol 7603. Springer, Berlin, Heidelberg
- Deloit C (2014) Publishing the British national bibliography as linked open data. *Cat Index* 174: 13–18
- DPLA (2017) Digital public library of America (DPLA). Website. <https://dp.la/>. Accessed 12 Dec 2012
- GBHGIS (2012) Great Britain historical geographical information system (GBHGIS). <http://www.port.ac.uk/research/gbhgis/>. Accessed 12 Dec 2017
- Getty TGRI (2017) Getty vocabularies as linked open data. Online. <http://www.getty.edu/research/tools/vocabularies/lod/index.html>. Accessed 07 Dec 2017
- Haslhofer B, Isaac A (2011) data.europeana.eu: the Europeana linked open data pilot. In: International conference on Dublin core and metadata applications, pp 94–104
- Haslhofer B, Robitz W, Guimbretiere F, Lagoze C (2013) Semantic tagging on historical maps. In: Proceedings of the 5th annual ACM web science conference. ACM, pp 148–157
- Hitzler P, Krötsch M, Parsia B, Schneider PFP, Rudolph S (2012) Owl 2 web ontology language primer, 2nd edn. W3C recommendation. <https://www.w3.org/TR/owl2-primer/>. Accessed 07 Dec 2017
- Isaksen L, Simon R, Barker ET, de Soto Cañamares P (2014) Pelagios and the emerging graph of ancient world data. In: Proceedings of the 2014 ACM conference on web science (WebSci'14). ACM, New York, pp 197–201. <https://doi.org/10.1145/2615569.2615693>
- Malmsten M (2008) Making a library catalogue part of the semantic web. In: International conference on Dublin core and metadata applications-metadata for semantic and social applications (DC-2008), Berlin, 22–26 Sept 2008. Humboldt-Universität zu, Berlin
- Michel JB, Shen YK, Aiden AP, Veres A, Gray MK, Team TGB, Pickett JP, Holberg D, Clancy D, Norvig P, Orwant J, Pinker S, Nowak MA, Aiden EL (2010) Quantitative analysis of culture using millions of digitized books. *Science*. <http://www.sciencemag.org/content/331/6014/176.full>
- OCLC (2017) OCLC linked data. Online. <https://www.oclc.org/developer/develop/linked-data.en.html>. Accessed 07 Dec 2012
- PBW (2011) Prosopography of the Byzantine world. Online. <http://blog.pbw.cch.kcl.ac.uk/>. Accessed 12 Dec 2012
- Pleiades Contributors (2017) The Pleiades Gazetteer of the ancient world. <http://pleiades.stoa.org/>. Accessed 12 Dec 2012
- Schich M, Song C, Ahn YY, Mirsky A, Martino M, Barabasi AL, Helbing D (2014) A network framework of cultural history. *Science* 345(6196):558–562. <https://doi.org/10.1126/science.1240064>. Received for publication 6 May 2013. Accepted for publication 13 June 2014
- Siemens R, Schreibman S (2008) A companion to digital literary studies: blackwell companions to literature and culture. Wiley Publishing. <https://www.wiley.com/en-us/A+Companion+to+Digital+Literary+Studies-p-9781405148641>
- Simon R, Barker E, Isaksen L, de Soto Cañamares P (2015) Linking early geospatial documents, one place at a time: annotation of geographic documents with recogito. *e-Perimetron* 10(2):49–59. <http://oro.open.ac.uk/43613/>
- Singhal A (2012) Introducing the knowledge graph: things, not strings. <https://goo.gl/U168iz>. Accessed 21 Nov 2017
- Summers E, Isaac A, Redding C, Krech D (2008) LCSH, SKOS and linked data. In: International conference on Dublin core and metadata applications-metadata for semantic and social applications (DC-2008), Berlin, 22–26 Sept 2008. Humboldt-Universität zu, Berlin
- TIB (2017) Digital Tabula Imperii Byzantini. Online. <http://tib.oewa.ac.at/>. Accessed 12 Dec 2017
- TIR-FOR (2016) Tabula Imperii Romani – Forma Orbis Romani. <http://tir-for.iec.cat/>. Accessed 12 Dec 2017
- ToL (2017) The treasury of lives – a biographical encyclopedia of Tibet, inner Asia, and the Himalaya. Online. <http://treasuryoflives.org/>. Accessed 12 Dec 2017
- Vrandečić D, Krötsch M (2014) Wikidata: a free collaborative knowledgebase. *Commun ACM* 57(10):78–85
- W3C (2011) Library linked data incubator group final report. Online. <https://www.w3.org/2005/Incubator/ld/>. Accessed 13 Dec 2017



Lambda Architecture

► Architectures

Languages for Big Data Analysis

Marco Aldinucci¹, Maurizio D’Rocco¹,
Claudia Misale², and Guy Tremblay³

¹Computer Science Department, University of Turin, Turin, Italy

²Cognitive and Cloud, Data-Centric Solutions, IBM T.J. Watson Research Center, New York, NY, USA

³Département d’Informatique, Université du Québec à Montréal, Montréal, QC, Canada

Definitions

In this chapter, some of the most common tools for Big Data analytics are surveyed, inter-alia, Apache Spark, Flink, Storm, and Beam. They are compared against well-defined features concerning programming model (language expressivity and semantics), and execution model (parallel behaviour and run-time support). The implementation of a running example is provided for all of them.

Overview

Boosted by Big Data popularity, new languages and frameworks for data analytics are appearing at an increasing pace. Each of them introduces its own concepts and terminology and advocates a (real or alleged) superiority in terms of performances or expressiveness against predecessors. In this hype, for a user approaching Big Data analytics (even an educated computer scientist), it might be difficult to have a clear picture of the programming model underneath these tools and the expressiveness they provide to solve some user-defined problem.

To provide some order in the world of Big Data processing, a toolkit of models to identify their common features is introduced, starting from data layout.

Data processing applications are divided into *batch* vs. *stream* processing. Batch programs process one or more *finite* datasets to produce a resulting finite output dataset, whereas stream programs process possibly unbounded sequences of data, called *streams*, doing so in an incremental manner. Operations over streams may also have to respect a total data ordering – for instance, to represent time ordering.

In order to compare the expressiveness of programming models for Big Data, analytics are mapped onto a unifying (and lower-level) computation model, viz. the *dataflow model* (Lee and Parks 1995). As shown in Misale et al. (2017), the dataflow model is able to capture

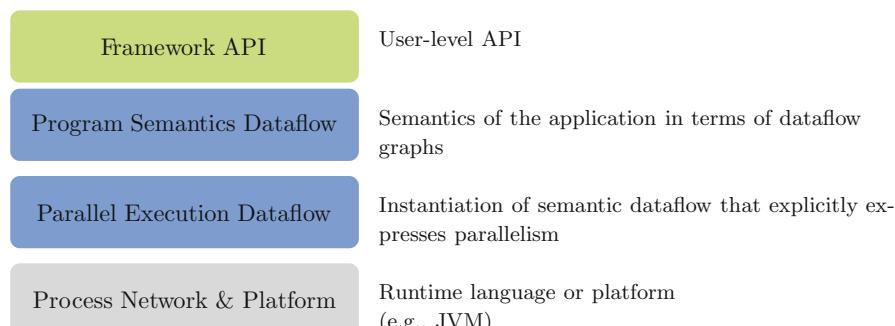
the distinctive features of all frameworks at all levels of abstraction, from the user-level API to the execution model. In the dataflow model, an application acts as a directed graph of actors. In its “modern” macro-data flow version (Aldinucci et al. 2012), it naturally models independent (thus parallelizable) kernels starting from a graph of true data dependencies, where a kernel’s execution is triggered by data availability.

The dataflow model is expressive enough to describe batch, micro-batch, and streaming models that are implemented in most tools for Big Data processing. Also, the dataflow model helps in maturing the awareness that many Big Data analytic tools share almost the same base concepts, differing mostly in their implementation choices.

In the next section, the key finding of this chapter is introduced, i.e., the layered dataflow model. On this ground, the mainstream languages and frameworks for Big Data analytics are then introduced; they are Spark, Storm, Flink, and Beam. According to the layered dataflow model, they will be discussed at two successive levels of abstractions, i.e., (1) *API and semantics* and (2) *parallel execution and runtime support*. This section draws future direction for research.

Key Research Finding: The Layered Dataflow Model

In order to compare different Big Data frameworks, a revised version of the layered dataflow model is adopted (Misale et al. 2017).



Languages for Big Data Analysis, Fig. 1 Layered model representing the levels of abstractions provided by the frameworks that were analyzed

This model, sketched in Fig. 1, presents four layers. The top layer captures the framework API. The two intermediate layers are dataflow models with different semantics, as described in the paragraphs below. The bottom layer is the *process network and platform*, which embodies the network of processes used to implement a given framework together with their programming language runtime support (e.g., Java and Scala in Spark), a level which is beyond the scope of this work.

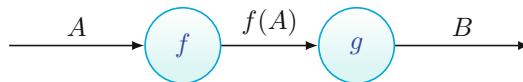
The stacked dataflow layers are as follows.

Framework API

At the top of the stack, the API is the concrete primitives provided by each framework, in which data processing applications are written. Within the Big Data analytic domain, APIs can be regarded as domain-specific languages (DSLs), expressed in some host language (e.g., Python, Scala, Java).

Program Semantic Dataflow

The API exposed by all major Big Data frameworks can be explained in terms of a dataflow graph, that is, a pair $G = \langle V, E \rangle$ where actors V represent operators, channels E represent data dependencies among operators, and tokens represent data to be processed. For instance, consider to process a collection A by a function f followed by a function g . This is represented by the graph in Fig. 2, which represents the semantic dataflow of a program computing the functional composition $f \circ g$.



Languages for Big Data Analysis, Fig. 2 Semantic dataflow graph for $f \circ g$, expressing data dependencies

Moreover, although not extensively discussed here, it is remarkable that also nonfunctional aspects (e.g., window-based stream processing, stateful operators, iterations) can be conveniently represented by means of the proposed dataflow setting (Misale et al. 2017).

Parallel Execution Dataflow

This level represents an instantiation of the semantic dataflows in terms of processing elements (i.e., actors) connected by data channels (i.e., edges). The most straightforward source of parallelism comes directly from the dataflow model, namely, independent actors can run in parallel. Furthermore, some actors can be replicated to increase parallelism by making replica work over a *partition* of the input data – that is, by exploiting full *data parallelism*. Both the above schemas are referred as *embarrassingly parallel* processing, since there are no dependencies among actors. Finally, in case of dependent actors that are activated multiple times (e.g., by further splitting a partition into multiple tokens), parallelism can still be exploited by letting tokens “flow” as soon as each activation is completed. This well-known schema is referred to as *stream/pipeline* parallelism.

Process Network Dataflow and Platform

This level describes how the program is effectively deployed and executed onto the underlying platform. Actors are concrete computing entities (e.g., processes), and edges are communication channels. The most common approach is for the actual network to be a master-worker task executor.

Examples of Application

In this section, some mainstream frameworks for Big Data processing are discussed, within the layered dataflow setting. “Google MapRe-

duce” is commonly considered as the first widespread tool in this domain. “► [Apache Spark](#)” provides a richer API (e.g., also targeting streams) and an implementation optimized for iterative processing. “► [Apache Flink](#)” is similar to Spark, but it exploits more parallelism by means of a stream-based runtime. “Apache Storm” is focused on stream processing, and differently from the previous frameworks, Storm programs are defined as interconnected processing units, rather than functional compositions. Finally, “Apache Beam” provides an alternative API for unifying batch and stream processing under a single programming model.

Google MapReduce

Google can be considered the pioneer of Big Data processing, as the publication of the MapReduce framework paper (Dean and Ghemawat 2004) made this model mainstream. Based on the *map* and *reduce* functions, commonly used in parallel and functional programming (Cole 1989), MapReduce provides a native key-value model and built-in sorting, which made it successful for several Big Data analytics scenarios.

API

A MapReduce program is built on the following user-defined functions: (1) a map function, which is independently applied to each item from an input key-value dataset to produce an *intermediate* key-value dataset; (2) a reduce function, which combines all the intermediate values associated with each key (together with the key itself) into lists of reduced values (one per key); and (3) a partitioner function, which is used while *sorting* the intermediate dataset (i.e., before being reduced), so that the order over the key space is respected within each partition identified by the partitioner.

Figure 3 shows a source code extract from a MapReduce implementation of the Word Count application that counts the occurrences of each word in an input text and is generally considered as the “Hello World!” for Big Data analytics. In the code extract, only map and reduce functions are specified; thus, a default implementation-dependent sorting is used.

```

1 public class WordCount {
2
3     public static class TokenizerMapper
4             extends Mapper<Object,Text,Text,IntWritable> {
5         private final static IntWritable one = new IntWritable(1);
6         private Text word = new Text();
7
8         public void map(Object key, Text value, Context context)
9                 throws IOException, InterruptedException {
10            StringTokenizer itr = new StringTokenizer(value.toString());
11            while (itr.hasMoreTokens()) {
12                word.set(itr.nextToken());
13                context.write(word, one);
14            }
15        }
16    }
17
18    public static class IntSumReducer
19            extends Reducer<Text,IntWritable,Text,IntWritable> {
20        private IntWritable result = new IntWritable();
21
22        public void reduce(Text key, Iterable<IntWritable> values, Context
23        context)
24                throws IOException, InterruptedException {
25            int sum=0 ;
26            for (IntWritable val: values) {
27                sum += val.get();
28            }
29            result.set(sum);
30            context.write(key, result);
31        }
32    }
33 }

```

Languages for Big Data Analysis, Fig. 3 The map and reduce functions for a Java Word Count class example in MapReduce

Semantics

The semantics of MapReduce is defined by the following chain of higher-order operators, where f , h , and \oplus correspond to the map, partitioner, and reduce API functions, respectively:

$$\begin{aligned} \text{map-reduce } f \ h \ R \oplus \\ = (\text{flat-map } f) \circ (\text{sort } h \ R) \circ (\text{reduce } \oplus) \end{aligned}$$

When applied to an input multiset (i.e., a finite unordered collection, possibly containing duplicated items) of key-value pairs, flatMap applies the kernel to each item and collapses all the results into a single intermediate multiset. Formally, $f : K \times V \rightarrow \mathcal{P}(K' \times V')$ is the kernel and m is a collection with $(K \times V)$ -typed items:

$$\text{flat-map } f \ m = \cup \{f(k, v) : (k, v) \in m\} \quad (k', v') \in m'$$

The intermediate multiset is processed by the sort operator to produce a multiset of partially ordered multisets (called intermediate *partitions* in MapReduce terminology). This partial sorting is parametric with respect to the number of partitions and the partitioning function, the latter mapping key-value input pairs to a partition. Formally, where I is the partition identifier space, R is the number of partitions, and $h: K' \times \mathbb{N} \rightarrow P$ is the partitioning function (e.g., hash-based), the partition identified by $\iota \in I$ from the intermediate multiset m' is defined as:

$$\sigma_\iota^h(R, m') = \{(k', v') : h(k', R) = \iota,$$

Moreover, each partition p is partially ordered according to the keys, such that:

$$\begin{aligned} \forall (k'_a, v'_a), (k'_b, v'_b) \in p, k'_a < k'_b \\ \Rightarrow (k'_a, v'_a) \text{ precedes } (k'_b, v'_b) \text{ in } p \end{aligned}$$

However, no ordering is guaranteed “internally” to each given key. Then, the semantics of sort follows:

$$\text{sort } h R m' = \left\{ \sigma_i^h(R, m') : \sigma_i^h(R, m') \neq \emptyset \right\}$$

Finally, the reduce operator synthesizes the partitions on a per-key basis, according to a reduction kernel, respecting the ordering within each partition.

First, the reduce-by-key operator is defined. Given a collection of key-value pairs, it produces a collection of synthesized collections, one for each key. Formally, given $\oplus : K' \times P(V') \rightarrow P(V'')$ denoting the reduction kernel:

$$\begin{aligned} \text{reduce-by-key } \oplus p = \\ \{(k', \oplus(k', \{v' : (k', v') \in p\})) : \exists (k', v') \in p\} \end{aligned}$$

Moreover, when applied to partitions (i.e., multisets partially ordered based on a per-key base), reduce-by-key operator produces a *totally ordered set*, where all the values with a given key are combined into a single key-value(s) pair and the ordering is kept among keys. Then, the reduce operator is defined as follows, where m'_s is a multiset of partitions and the big union operator respects the ordering:

$$\text{reduce } \oplus m'_s = \cup \{\text{reduce-by-key } \oplus p : p \in m'_s\}$$

Parallel Execution

A simple form of data parallelism can be exploited on the flatMap side, by partitioning the input collection into n chunks and having n executors process a chunk. In dataflow terms, this corresponds to a graph with n actors, each processing a token that represents a chunk. Each flatMap executor emits R (i.e., the number of intermediate partitions) chunks, each containing

the intermediate key-value pairs mapped to a given partition.

The intermediate chunks are processed by R reduce executors. Each executor (1) receives n chunks (one from each flatMap executor); (2) merges the chunks into an intermediate partition and partially sorts it based on keys, as discussed above; and (3) performs the reduction on a per-key basis. Finally, a downstream collector gathers R tokens from the reduce executors and merges them into the final result.

A key aspect in the depicted parallelization is the *shuffle* phase, in which data is distributed between flatMap and reduce operators, according to an *all-to-all* communication schema. This poses severe challenges from the implementation standpoint.

Runtime Support

The most widespread implementation (i.e., *Hadoop*) is based in a *master-worker* approach, in which the master retains the control over the global state of the computation and informs the workers about the tasks to be executed.

A cornerstone of Hadoop is its distributed file system (HDFS), which is used to exchange data among workers, in particular upon shuffling. As a key feature, HDFS exposes the *locality* for stored data, thus enabling the principle of moving the computation toward the data, to minimize the communication. However, disk-based communication leads to performance problems when dealing with iterative computations, such as machine learning algorithms (Chu et al. 2006).

Apache Spark

Apache Spark (Zaharia et al. 2012) was proposed to overcome some limitations in Google MapReduce. Instead of a fixed processing schema, Spark allows datasets to be processed by means of arbitrarily composed primitives, referred to as the application directed acyclic graph (DAG). Moreover, instead of relying exclusively on disks for communicating data among the processing units, in-memory caching is exploited to boost the performance, in particular for iterative processing.

API and Semantics

Apache Spark uses a *declarative processing style* expressed as methods on objects representing collections. More precisely, Apache Spark implements batch programming with a set of operators, called *transformations*, that are uniformly applied to whole datasets called *resilient distributed datasets* (RDD) (Zaharia et al. 2012), which are immutable multisets. A Spark program can be characterized by the two kinds of operations applicable to RDDs: *transformations* and *actions*. Transformations are lazy functions (i.e., they do not compute their results right away) over collections – such as map, reduce, and flatMap – that are uniformly applied to whole RDDs. Actions effectively trigger the DAG execution, and they return a value to the user.

Listing 4 shows the source code for a simple Word Count application in the Java Spark API.

For stream processing, Spark implements an extension through the Spark Streaming

module, providing a high-level abstraction called *discretized stream* or *DStream*. Such streams represent results in continuous sequences of RDDs of the same type, called *micro-batches*. Operations over DStreams are “forwarded” to each RDD in the DStream; thus, the semantics of operations over streams is defined in terms of batch processing according to the simple translation $\text{op}(a) = [\text{op}(a_1), \text{op}(a_2), \dots]$, where $[\cdot]$ refers to a possibly unbounded ordered sequence, $a = [a_1, a_2, \dots]$ is a DStream, and each item a_i is a micro-batch of type RDD.

When mapping a Spark program to a semantic graph, tokens represent RDDs and DStreams for batch and stream processing, respectively. Actors are operators – either transformations or actions – that transform data or return values (in-memory collection or files). Actors are activated only once in both batch and stream processing, since each collection (either RDD or DStreams) is represented by a single token (Fig. 4).

```

1 JavaRDD<String> textFile=sc.textFile("hdfs://...");  

2  

3 JavaRDD<String> words =  

4     textFile.flatMap(new FlatMapFunction<String, String>() {  

5         public Iterable<String> call(String s) {  

6             return Arrays.asList(s.split(" "));  

7         }  

8     });  

9  

10 JavaPairRDD<String, Integer> pairs =  

11     words.mapToPair(new PairFunction<String, String, Integer>() {  

12         public Tuple2<String, Integer> call(String s) {  

13             return new Tuple2<String, Integer>(s, 1);  

14         }  

15     });  

16  

17 JavaPairRDD<String, Integer> counts =  

18     pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {  

19         public Integer call(Integer a, Integer b) {  

20             return a + b;  

21         }  

22     });  

23  

24 counts.saveAsTextFile("hdfs://...");
```

Languages for Big Data Analysis, Fig. 4 A Java Word Count example in Spark from its example suite

Parallel Execution and Runtime Support

From the application DAG, Spark infers a parallel execution dataflow, in which many parallel instances of actors are created for each function and independent actors are grouped into *stages*. Due to the Spark batch-oriented implementation, each stage that depends on some previous stages has to wait for their completion before execution, according to the classical Bulk Synchronous Parallelism (BSP) approach. Therefore, a computation proceeds in a series of global *supersteps*, each consisting of (1) concurrent computation, in which each actor processes its own partition; (2) communication, where actors exchange data between themselves if necessary (the *shuffle* phase); and (3) barrier synchronization, where actors wait until all the other actors have reached the same barrier.

Similarly to the MapReduce implementation, Spark's execution model relies on the master-worker model: a cluster manager (e.g., YARN) manages resources and supervises the execution of the program. It manages application scheduling to worker nodes, which execute the application logic (the DAG) that has been serialized and sent by the master.

Apache Flink

Apache Flink (Carbone et al. 2015) is similar to Spark, in particular from the API standpoint. However, Flink is based on *streaming* as a primary concept, rather than a mere linguistic extension on top of batch processing (as is in Spark). With the exception of iterative processing, stream parallelism is exploited to avoid expensive synchronizations among successive phases, when executing both batch and stream programs.

API and Semantics

Apache Flink's main focus is on stream programming. The abstraction used is the *DataStream*, which is a representation of a stream as a single object. Operations are composed (i.e., pipelined) by calling operators on *DataStream* objects. Flink also provides the *dataset* type for batch applications that identifies a single immutable multiset – a stream of one element. A Flink program, either for stream or batch processing, is a term from an algebra of operators over *DataStream*s or *dataset*s, respectively.

Figure 5 shows Flink's source code for the simple Word Count application.

```

1 public class WordCountExample {
2     public static void main(String[] args) throws Exception {
3         final ExecutionEnvironment env =
4             ExecutionEnvironment.getExecutionEnvironment();
5         DataSet<String> text =
6             env.fromElements("Text...\"");
7         DataSet<Tuple2<String, Integer>> wordCounts =
8             text.flatMap(new LineSplitter())
9                 .groupBy(0)
10                .sum(1);
11
12         wordCounts.print();
13     }
14
15     public static class LineSplitter
16         implements FlatMapFunction<String, Tuple2<String, Integer>> {
17         @Override
18         public void flatMap(String line, Collector<Tuple2<String, Integer>> out) {
19             for (String word : line.split(" ")) {
20                 out.collect(new Tuple2<String, Integer>(word, 1));
21             }
22         }
23     }
24 }
```

Languages for Big Data Analysis, Fig. 5 A Java Word Count example in Flink from its example suite

Flink applications can be applied to semantic dataflow graphs in the same way as for Spark, by treating datasets and datastreams as, respectively, RDDs and DStreams.

Parallel Execution and Runtime Support

Flink transforms a JobGraph into an execution graph, in which the JobVertex contains ExecutionVertexes (actors), one per parallel sub-task. A key difference compared to the Spark execution graph is that, apart from iterative processing (that is still executed under BSP), there is no barrier among actors or vertexes: instead, there is effective pipelining.

Also Flink's execution model relies on the master-worker model: a deployment has at least one job manager process that coordinates checkpointing and recovery and that receives Flink jobs. The job manager also schedules work across the task manager processes (i.e., workers) which usually reside on separate machines and in turn execute the code.

Apache Storm

Apache Storm (Nasir et al. 2015; Toshniwal et al. 2014) is a framework targeting only stream processing. It is perhaps the first widely used large-scale stream processing framework in the open source world. Whereas Spark and Flink are based on a declarative data processing model – i.e., they provide as building blocks for data collections and operations on those collections – Storm is instead based on a “topological” approach in that it provides an API to explicitly build graphs.

API and Semantics

Storm's programming model is based on three key notions: *spouts*, *bolts*, and *topologies*. A spout is a source of a stream that is (typically) connected to a data source or that can generate its own stream. A bolt is a processing element, so it processes any number of input streams and produces any number of new output streams. Most of the logic of a computation goes into bolts, such as functions, filters, streaming joins, or streaming aggregations. A topology is the composition of spouts and bolts resulting in a network.

Storm uses *tuples* as its data model, that is, named lists of values of arbitrary type. Hence, bolts are parametrized with per-tuple kernel code. It is also possible to define cyclic graphs by way of feedback channels connecting bolts. Figure 6 shows Storm's source code for the simple Word Count application.

As for mapping applications to dataflow graphs, whereas in the previously described frameworks the graph is implicit and tokens represent whole datasets or streams, Storm is different: (1) the dataflow graph is already explicit, as it is constructed using the provided API; (2) each token represents a single stream item (tuple), and the actors, representing (macro) dataflow operators, are activated each time a new token is available.

Parallel Execution and Runtime Support

At execution level, each actor is replicated to increase the inter-actor parallelism, and each group of replicas corresponds to the bolt/spout in the semantic dataflow.

Each of these actors represents independent data parallel tasks, on which pipeline parallelism is exploited. Eventually, tasks are executed by a master-worker engine, as in the previous frameworks.

Google Cloud Platform and Apache Beam

Google Dataflow SDK (Akidau et al. 2015) is part of the Google Cloud Platform. Google Dataflow supports a simplified pipeline development via Java and Python APIs in the Apache Beam SDK, which provides a set of windowing and session analysis primitives as well as an ecosystem of source and sink connectors. Apache Beam allows the user to create pipelines that are then executed by one of Beam's supported distributed processing back ends, which include, among others, Apache Flink, Apache Spark, and Google Cloud Dataflow, which are called *runners*.

API and Semantics

Apache Beam programming model is centered around the concept of *pipeline*, which represents a data processing program consisting of

```

1  public class WordCountTopology {
2      public static class SplitSentence extends ShellBolt implements IRichBolt {
3          public SplitSentence() {
4              super("python", "splitsentence.py");
5          }
6
7          @Override
8          public void declareOutputFields(OutputFieldsDeclarer declarer) {
9              declarer.declare(new Fields("word"));
10         }
11
12         @Override
13         public Map<String, Object> getComponentConfiguration() {
14             return null;
15         }
16     }
17
18     public static class WordCount extends BaseBasicBolt {
19         Map<String, Integer> counts = new HashMap<String, Integer>();
20
21         @Override
22         public void execute(Tuple tuple, BasicOutputCollector collector) {
23             String word = tuple.getString(0);
24             Integer count = counts.get(word);
25             if (count == null) count = 0;
26             count++;
27             counts.put(word, count);
28             collector.emit(new Values(word, count));
29         }
30
31         @Override
32         public void declareOutputFields(OutputFieldsDeclarer declarer) {
33             declarer.declare(new Fields("word", "count"));
34         }
35     }
36
37     public static void main(String[] args) throws Exception {
38         TopologyBuilder builder = new TopologyBuilder();
39         builder.setSpout("spout",
40             new RandomSentenceSpout(), 5);
41         builder.setBolt("split",
42             new SplitSentence(), 8).shuffleGrouping("spout");
43         builder.setBolt("count",
44             new WordCount(), 12).fieldsGrouping("split", new
45             Fields("word"));
46
47         Config conf = new Config();
48         conf.setDebug(true);
49         conf.setNumWorkers(3);
50
51         StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
52     }
53 }
```

Languages for Big Data Analysis, Fig. 6 A Java Word Count example in Storm from its example suite

```

1 static class ExtractWordsFn extends DoFn<String, String> {
2     private final Counter emptyLines =
3         Metrics.counter(ExtractWordsFn.class, "emptyLines");
4     private final Distribution lineLenDist =
5         Metrics.distribution(ExtractWordsFn.class, "lineLenDistro");
6
7     @ProcessElement
8     public void processElement(ProcessContext c) {
9         lineLenDist.update(c.element().length());
10        if (c.element().trim().isEmpty())
11            emptyLines.inc();
12
13        // Split the line into words.
14        String[] words = c.element().split(ExampleUtils.TOKENIZER_PATTERN);
15
16        // Output each word encountered into the output PCollection.
17        for (String word : words)
18            if (!word.isEmpty())
19                c.output(word);
20    }
21
22    public static class FormatAsTextFn
23        extends SimpleFunction<KV<String, Long>, String> {
24        @Override
25        public String apply(KV<String, Long> input) {
26            return input.getKey() + ": " + input.getValue();
27        }
28    }
29
30    public static class CountWords
31        extends PTransform<PCollection<String>, PCollection<KV<String, Long>>> {
32        @Override
33        public PCollection<KV<String, Long>> expand(PCollection<String> lines) {
34            // Convert lines of text into individual words.
35            PCollection<String> words =
36                lines.apply(ParDo.of(new ExtractWordsFn()));
37
38            // Count the number of times each word occurs.
39            PCollection<KV<String, Long>> wordCounts =
40                words.apply(Count.<String>perElement());
41
42            return wordCounts;
43        }
44    }
45
46    public static void main(String[] args) {
47        // options initialization ...
48        Pipeline p = Pipeline.create(options);
49
50        p.apply("ReadLines", TextIO.read().from(options.getInputFile()))
51            .apply(new CountWords())
52            .apply(MapElements.via(new FormatAsTextFn()))
53            .apply("WriteCounts", TextIO.write().to(options.getOutput()));
54
55        p.run().waitUntilFinish();
56    }
57 }

```

Languages for Big Data Analysis, Fig. 7 Java code fragment for a Word Count example in Apache Beam from its example suite

a set of operations that can read a source of input data, transform that data, and write out the resulting output. A pipeline can be linear, but it can also branch and merge, thus making a pipeline a DAG defined through conditionals, loops, and other common programming structures. A pipeline stage is a *transform*, which accepts one or more *PCollections* (i.e., a possibly unbounded immutable collection, either ordered or not) as input, performs an operation on its elements, and produces one or more new PCollections as output. The *ParDo* is the core element-wise transform in Apache Beam, invoking a user-specified function on each of the elements of the input PCollection to produce zero or more output elements (*flatMap* semantics) collected into an output PCollection.

When mapping a Beam program into a semantic graph, tokens represent PCollections and actors are transforms accepting PCollections in input and producing PCollections in output.

Figure 7 shows how to create a Word Count pipeline.

Parallel Execution and Runtime Support

The bounded (or unbounded) nature of a PCollection also affects how data is processed. Bounded PCollections can be processed using batch jobs that might read the entire dataset once and perform processing as a finite job. Unbounded PCollections must be processed using streaming jobs – as the entire collection will never be available for processing at any one time – and bounded subcollections can be obtained through logical finite size windows.

As mentioned, Beam relies on the *runner* specified by the user. When executed, an entity called *Beam Pipeline Runner* (related to execution back end) translates the data processing pipeline into the API compatible with the selected distributed processing back end. Hence, it creates an execution graph from the pipeline, including all the transforms and processing functions. That graph is then executed using the appropriate distributed processing back end, becoming an asynchronous job/process on that back end; thus, the final parallel execution graph is generated by the back end.

The parallel execution dataflow is similar to the one in Spark and Flink, and parallelism is expressed in terms of data parallelism in transforms (e.g., *ParDo* function) and inter-actor parallelism on independent transforms. In Beam's nomenclature, this graph is called the execution graph. Similarly to Flink, pipeline parallelism is exploited among successive actors.

Future Direction for Research

In this chapter, some of the most common tools for analytics and data management were presented. One common drawback of all of them is that their operators are not polymorphic with respect to the data model (e.g., stream, batch). This means analytics pipelines should be either re-designed or simulated to match a different data model, as in the *Lambda* or *kappa* architectures, respectively. This is often hardly acceptable either in terms of effort or performance.

The PiCo framework has been recently proposed to overcome this problem. PiCo's programming model aims at making easier the programming of data analytics applications while preserving or enhancing their performance. This is attained through three key design choices: (1) unifying batch and stream data access models, (2) decoupling processing from data layout, and (3) exploiting a stream-oriented, scalable, efficient C++11 runtime system. PiCo proposes a programming model based on pipelines and operators that are polymorphic with respect to data types in the sense that it is possible to reuse the same algorithms and pipelines on different data models (e.g., streams, lists, sets, etc.). Being PiCo designed as a C++11 header-only library, it can be easily ported in any general-purpose or specialized device supporting vanilla C++ runtime, including any low-power device in the edge of computing. Preliminary results show that PiCo can attain equal or better performances in terms of execution times and hugely improve memory utilization when compared to Spark and Flink in both batch and stream processing (Misale 2017; Misale et al. 2018).

Cross-References

- ▶ [Apache Flink](#)
- ▶ [Apache Spark](#)
- ▶ [Big Data Analysis Techniques](#)
- ▶ [Big Data Architectures](#)
- ▶ [BSP Programming Model](#)
- ▶ [Hadoop](#)
- ▶ [Orchestration Tools for Big Data](#)
- ▶ [Parallel Processing with Big Data](#)
- ▶ [Performance Evaluation of Big Data Analysis](#)
- ▶ [Scalable architectures for Big Data Analysis](#)
- ▶ [Tools and libraries for Big Data Analysis](#)

References

- Akida T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E, Whittle S (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc VLDB Endowment 8:1792–1803
- Aldinucci M, Danelutto M, Anardu L, Torquati M, Kilpatrick P (2012) Parallel patterns + macro data flow for multi-core programming. In: Proceedings of International Euromicro PDP 2012: parallel distributed and network-based processing. IEEE, Garching, pp 27–36
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flinkTM: Stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38
- Chu CT, Kim SK, Lin YA, Yu Y, Bradski G, Ng AY, Olukotun K (2006) Map-reduce for machine learning on multicore. In: Proceedings of the 19th International conference on Neural information processing systems, Vancouver, pp 281–288
- Cole M (1988) Algorithmic skeletons: a structured approach to the management of parallel computation. PhD thesis, University of Edinburgh
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of 6th Usenix symposium on operating systems design & implementation, San Francisco, pp 137–150
- Lee EA, Parks TM (1995) Dataflow process networks. Proc IEEE 83(5):773–801
- Misale C (2017) PiCo: a domain-specific language for data analytics pipelines. PhD thesis, Computer Science Department, University of Torino
- Misale C, Drocco M, Aldinucci M, Tremblay G (2017) A comparison of big data frameworks on a layered dataflow model. Parallel Process Lett 27(01):1740003
- Misale C, Drocco M, Tremblay G, Aldinucci M (2018) PiCo: a novel approach to stream data analytics. In:

Proceedings of Euro-Par workshops: 1st international workshop on autonomic solutions for parallel and distributed data stream processing (Auto-DaSP 2017), Santiago de Compostela. LNCS, vol 10659

Nasir MAU, Morales GDF, García-Soriano D, Kourtellis N, Serafini M (2015) The power of both choices: practical load balancing for distributed stream processing engines. In: 2015 IEEE 31st international conference on data engineering, Seoul, pp 137–148

Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm@twitter. In: Proceedings of the ACM SIGMOD international conference on management of data, Snowbird, pp 147–156

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation, Vancouver

Large-Scale Entity Resolution

Erhard Rahm and Eric Peukert
University of Leipzig, Leipzig, Germany

Synonyms

[Data deduplication](#); [Link discovery](#); [Object matching](#); [Record linkage](#)

Definitions

The goal of entity resolution is the identification of semantically equivalent objects within one data source or between different sources. In the context of Big Data, there is a growing need for large-scale entity resolution to find matching entities within very large and between many data sources. This requires effectively parallelizing entity resolution tasks within cluster environments.

Overview

Entity resolution (ER) is the task to identify semantically equivalent entities referring to the

same real-word object (e.g., persons, products, publications, or movies) within one data source or between different sources. This task is also known as data deduplication, object matching, record linkage, or link discovery. ER is of core importance for data cleaning and data integration and has been addressed for a long time in practice and research (Rahm and Do 2000; Elmagarmid et al. 2007; Christen 2012).

The traditional focus of ER was on structured data in relational databases. The example in Fig. 1 shows two input relations from different sources with address data that contain duplicates. The example illustrates some of the challenges of data deduplication tasks such as heterogeneous attribute names and value differences due to different conventions (e.g., for gender), abbreviations, omissions, nick names, and errors in the data. Typically, ER requires to first determine comparable attributes and then computing and combining the similarity for multiple attributes, e.g., for name, gender, and address in the example.

Large-scale entity resolution for Big Data introduces a number of challenges:

- Data sources can be very large with many millions of entities making it hard to achieve both effective and efficient entity resolution. Effectiveness asks for a good match quality so that only truly matching entities are identified (good precision) and that all matches are found (good recall). Efficiency is a problem since it is not feasible to pairwise compare all entities with each other for large datasets. Hence, it is imperative to reduce the number of comparisons (by filtering and blocking

techniques) and to also apply parallel entity resolution on many processors.

- Relevant data can be spread over thousands of data sources so that determining pairwise matches between two sources is not sufficient. Rather, a more holistic entity resolution is necessary in such cases where matching entities of all sources are grouped or clustered together such that all entities in such clusters match with each other (Rahm 2016). An example for such large-scale entity resolution is product offers from thousands of web shops that should be matched with each other, e.g., to allow a price comparisons (Köpcke et al. 2012).
- Entities especially from the web or from social networks are often only semi-structured and contain free text and also image content. Data quality is further reduced due to the frequent use of heterogeneous names and abbreviations as well as missing values.
- Data sources can change quickly such that existing entities are changed and deleted and new entities are added. As a result, entity resolution should be an incremental process such that previously computed matches are retained and the match result is only updated for changed or new entities.

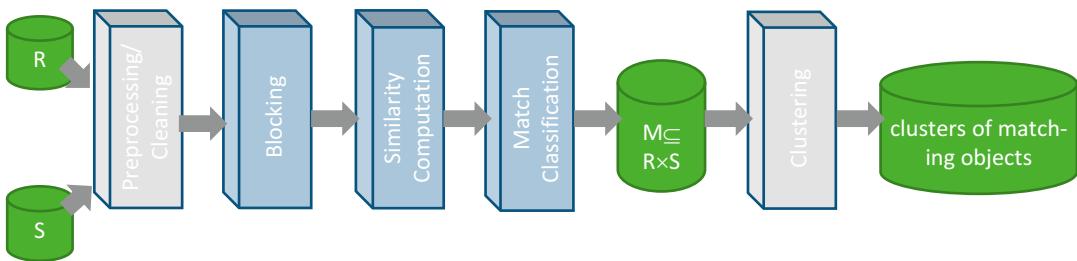
L

Entity resolution is typically implemented as a complex workflow to identify the matches within one data source or in several data sources. The output is either a set of pairwise correspondences (links) of matching entities, also called mapping. Alternatively all matching entities can be grouped or clustered together which is especially useful for more than two data sources. Figure 2

Cno	LastName	FirstName	Gender	Address	Phone/Fax
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

CID	Name	Street	City	Sex
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Large-Scale Entity Resolution, Fig. 1 Exemplary entity resolution task (from (Rahm and Do 2000))



Large-Scale Entity Resolution, Fig. 2 Entity resolution workflow for two input sources R and S

shows the main steps of typical ER workflows that include preprocessing/data cleaning, blocking, similarity computation, and match classification (Christen 2012; Elmagarmid et al. 2007; Dong and Srivastava 2015). Optionally, the results of pairwise matching can be clustered to group all directly or indirectly matching entities.

In the **preprocessing** phase, missing attributes can be computed or values are cleaned with the help of background knowledge. Textual content can be harmonized by removing capital letters, replacing special characters, tokenizing textual values, or applying NLP techniques such as stemming or stop word removal. Furthermore, names or keywords relevant for matching (so-called features) can be extracted from textual attributes for further processing. For example, matching product offers can benefit from extracting manufacturer information and so-called product codes from the product descriptions for later similarity computation (Köpcke et al. 2012).

The **blocking** phase is crucial to scale ER to large datasets to reduce the number of match comparisons. Similar entities are grouped within blocks, e.g., based on approaches such as Standard Blocking or Sorted Neighborhood (Christen 2012). With Standard Blocking entities are grouped into partitions or blocks based on a blocking function on the values of one or more attributes. The subsequent similarity computation is restricted to the pairwise matching of entities from the same block. For product offers one could use manufacturer as a blocking key so that only offers for products of the same manufacturer need to be compared with each other. This significantly reduces the search space but often reduces recall. Multipass blocking applies multiple block-

ing functions to reduce the loss on recall at the expense of more comparisons (Christen 2012).

The **similarity computation** phase computes pairwise similarities based on attribute level matching with domain-specific or general similarity functions, e.g., utilizing string similarity metrics (edit distance, n-gram, TF/IDF, etc.). Moreover, context-based matchers are applicable that incorporate the entity neighborhood for similarity computation, e.g., related products such as the cameras for which an accessory product such as a replacement battery can be used.

In the **match classification** phase, the computed entity similarities are used to decide whether a pair of entities matches or not. The classification can be expressed as rules based on a weighted combination of similarity values and a similarity threshold or it can be derived from a machine learning-based classifier that has been determined for suitable training data. The match candidates can be further postprocessed or filtered, e.g., to only consider the best matches if there are several match candidates per entity. For clean datasets without duplicates, there should be at most one matching entity in another dataset.

In the **clustering** phase, the computed matches (correspondences) can be used to group all directly and indirectly matching entities. The simplest approach of Connected Components computes a transitive closure of all correspondences. It can achieve a high match recall but often suffers from poor precision since a single wrong link can lead to large clusters. More sophisticated approaches such as correlation clustering (Pan et al. 2015) try to maximize the entity similarities within clusters and to minimize

similarities between clusters. Hassanzadeh et al. (2009) and Saeedi et al. (2017) comparatively evaluate several approaches for entity clustering.

These phases are implemented within many tools for entity resolution and link discovery as surveyed in (Köpcke and Rahm 2010; Christen 2012; Nentwig et al. 2017). The comparative evaluation of several tools in Köpcke et al. (2010) showed that learning-based match classification mostly achieves better match quality than rule-based approaches especially for more complex match tasks such as for product matching. Most of the previous tools, however, lack support for parallel processing making them insufficiently capable to meet the performance and scalability requirements of Big Data applications.

Key Research Findings

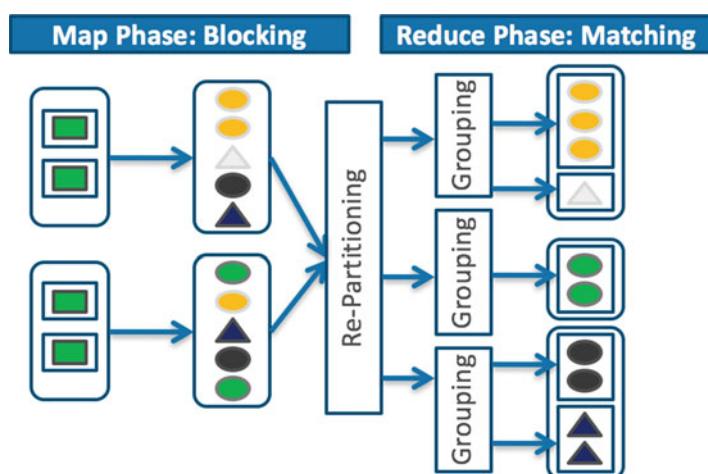
The performance and scalability requirements for Big Data applications require parallel entity resolution approaches in addition to the use of blocking. Kolb et al. (2012) have therefore investigated the utilization of Hadoop clusters and the MapReduce paradigm for parallel entity resolution. As illustrated in Fig. 3, there are two main phases. In the map phase, the input records are read in parallel and the blocking function is applied to the input data entities. The blocking key is used to distribute and group entities of the

same block to the same processing node for the matching phase. In the reduce phase, a pairwise matching is performed on all record pairs of the same block. Different blocks are processed in parallel.

Such a parallel entity resolution approach faces a potential load balancing problem in the reduce phase since the block sizes may vary to a large degree. The slowest reducer dominates the overall execution time and large blocks prevent the effective utilization of more than a few nodes. For load balancing, Kolb et al. (2012) propose two solutions called BlockSplit and PairRange. BlockSplit splits large blocks such there are multiple smaller match tasks that can be processed on different reducer nodes without creating overload situations. The approach leads to the replication of a subset of the entities that are part of several match tasks. The alternate PairRange approach globally enumerates all pairwise comparisons and evenly distributes these over all reducers. In both approaches, an additional MapReduce-Job is needed to compute the number and sizes of blocks for the configured blocking scheme. The load balancing approaches achieved a high scalability and proved to be stable against data skew. They are part of the DeDoop match tool that additionally offers machine-learning-based classification and a web-based UI for managing entity resolution tasks (Kolb and Rahm 2013).

Large-Scale Entity Resolution, Fig. 3

Parallel entity resolution with Map Reduce



Recently, alternative processing models such as Apache Flink and Apache Spark have become popular which provide a richer set of operators than MapReduce and in-memory computing for improved performance. Saeedi et al. ported many of the Dedoop features to Apache Flink within the FAMER system (Saeedi et al. 2017). In contrast to DeDoop, FAMER also supports multisource (holistic) entity resolution with more than two input sources. For this purpose, it offers parallel implementations for several entity clustering schemes.

Further proposals for holistic (multisource) entity resolution with entity clustering include Böhm et al. (2012), Pershina et al. (2015), and Nentwig et al. (2016). Gruenheid et al. (2014) studied incremental entity resolution that can improve scalability since new entities and sources are compared to existing clusters rather than with all other sources and entities.

Directions for Future Research

Novel machine learning techniques such as *deep learning* have been very successful in areas such as face or speech recognition and text mining. Such approaches are also promising for entity resolution, especially for web entities with textual descriptions, where deep neural networks may be able to automatically identify match-relevant features within such descriptions without extensive and complex preprocessing. Initial approaches in this direction are promising (Ebraheem et al. 2017) but do not yet fully exploit the potential of deep learning. Both the training of such methods and the model application need to be parallelized to be applicable for Big Data Integration tasks. The approaches should also be extended to include not only text attributes but also image content for matching.

Another area deserving more attention is scalable (parallel) entity resolution for graph-structured data where graphs (networks) can include entities and relationships of different types (e.g., publication and author entities with author and cite relationships). Here, the

similarity between entities of different graphs should consider not only the entity attributes but also the graph neighborhood. Furthermore, both entities (vertices) and relationships (edges) of the different types need to be matched and possibly fused within an integrated data graph. Moreover, holistic matching with more than two graphs and incremental entity resolution should be supported (Rahm 2016).

Another dimension that is beginning to gain interest is temporal record linkage that pays attention to the evolution of entities over time such as address changes (Li et al. 2011; Chiang et al. 2014; Christen et al. 2017). Such techniques should also become scalable to large and many data sources, e.g., within an incremental approach that keeps track of different entity versions.

References

- Böhm C, de Melo G, Naumann F, Weikum G (2012) LINDA: distributed Web-of- Data-scale entity matching. In: Proceedings of the conference on information and knowledge management, Maui, Hawaii
- Chiang YH, Doan A, Naughton JF (2014) Modeling entity evolution for temporal record matching. In: Proceedings of the ACM SIGMOD, Snowbird, Utah
- Christen P (2012) Data matching – concepts and techniques for record linkage, entity resolution, and duplicate detection, Springer
- Christen V, Groß A, Fisher J, Wang Q, Christen P, Rahm E (2017) Temporal group linkage and evolution analysis for census data. In: Proceedings of the extending database technology, Venice
- Dong XL, Srivastava D (2015) Big Data Integration. Morgan and Claypool, San Rafael
- Ebraheem M, Thirumuruganathan S, Joty S, Ouzzani M, Tang N (2017) DeepER – Deep entity resolution. CoRR abs/1710.00597
- Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: a survey. IEEE Trans Knowl Data Eng 19(1):1–16
- Gruenheid A, Dong XL, Srivastava D (2014) Incremental record linkage. Proc VLDB Endowment 7(9): 697–708
- Hassanzadeh O, Chiang F, Lee HC, Miller RJ (2009) Framework for evaluating clustering algorithms in duplicate detection. Proc VLDB Endowment 2(1): 1282–1293
- Kolb L, Rahm E (2013) Parallel entity resolution with Dedoop. Datenbank-Spektrum 13(1):23–32

- Kolb L, Thor A, Rahm E (2012) Load balancing for MapReduce-based entity resolution. In: Proceedings of the international conference on data engineering, Washington
- Köpcke H, Rahm E (2010) Frameworks for entity matching: a comparison. *Data Knowl Eng* 69(2):197–210
- Köpcke H, Thor A, Rahm E (2010) Evaluation of entity resolution approaches on real-world match problems. *Proc VLDB Endowment* 3(1–2):484–493
- Köpcke H, Thor A, Thomas S, Rahm E (2012) Tailoring entity resolution for matching product offers. In: Proceedings of the international conference on extending database technology, Berlin, pp 545–550
- Li P, Dong XL, Maurino A, Srivastava D (2011) Linking temporal records. *Proc VLDB Endowment* 4(11): 956–967
- Nentwig M, Groß A, Rahm E (2016) Holistic entity clustering for linked data. In: IEEE Data Mining Workshops (ICDMW), Barcelona
- Nentwig M, Hartung M, Ngonga Ngomo AC, Rahm E (2017) A survey of current link discovery frameworks. *Semantic Web* 8(3):419–436
- Pan X, Papailiopoulos D, Oymak S, Recht B, Ramchandran K, Jordan M (2015) Parallel correlation clustering on big graphs. In: Proceedings of the Advances in Neural Information Processing Systems, Montréal
- Pershina M, Yakout M, Chakrabarti K (2015) Holistic entity matching across knowledge graphs. In: Proceedings of the IEEE big data conference, Santa Clara
- Rahm E (2016) The case for holistic data integration. In: Proceedings of the advances in databases and information systems, Prague, Czech Republic, vol. 9809. Springer LNCS, Prague
- Rahm E, Do HH (2000) Data cleaning: problems and current approaches. In: IEEE data engineering bulletin
- Saeedi A, Peukert E, Rahm E (2017) Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In: Proceedings of the advances in databases and information systems, vol 10509. Springer LNCS, Nicosia

Large-Scale Graph Processing System

- ▶ [Graph Processing Frameworks](#)

Large-Scale Ontology Alignment

- ▶ [Large-Scale Schema Matching](#)

Large-Scale Schema Matching

Erhard Rahm and Eric Peukert
University of Leipzig, Leipzig, Germany

Synonyms

- ▶ [Large-scale ontology alignment](#)

Definitions

Schema matching aims at identifying semantically corresponding elements in two or more schemas, e.g., database schemas or ontologies. Large-scale schema matching focusses on the challenging cases of matching large schemas or more than two schemas. Matching multiple (>2) schemas is also known as holistic schema matching.

Overview

Schema matching aims at identifying semantic correspondences between metadata structures or models, such as database schemas, XML message formats, and ontologies. Solving such match problems is a key task in numerous application fields, in particular for data exchange and virtually all kinds of data integration. For example, in the two simple database schemas DB1.Student (Name, Major, Marks) and DB2.Grad-Student (LastName, FirstName, Major, Grades), simple (equivalence) correspondences would be DB1.Student \approx DB2.Grad-Student, DB1.Major \approx DB2.Major, and DB1.Marks \approx DB2.Grades, while DB1.Name is related to both DB2.LastName and DB2.FirstName (part-of correspondences “DB2.LastName – DB1.Name” and “DB2.FirstName – DB1.Name”). The inherent semantic heterogeneity in different schemas makes it difficult for automatic approaches to correctly determine the correspondences so that there is a need for user involvement. The complexity is especially

high for Big Data applications that frequently involve *large-scale schema matching* where large schemas with thousands of elements or many data sources with different schemas have to be dealt with. In this entry, we will focus on the pairwise matching of large schemas while matching of multiple schemas is treated under *Holistic schema matching*, sometimes also called *collective schema matching* (Rahm 2016).

Most approaches for schema matching focus on 2-way or pairwise matching where two related input schemas are matched with each other. As shown in Fig. 1, automatic matching may also make use of instance data for the input schemas or background knowledge such as dictionaries or previous match results that simplify the identification of corresponding elements in the input schemas. The result of pairwise schema matching is usually an equivalence *mapping* containing the identified semantic correspondences, i.e., pairs of semantically equivalent schema elements (e.g., schema attributes or ontology concepts). Some match approaches for ontologies also try to determine different kinds of semantic correspondences, such as is-a and part-of relationships between concepts (Euzenat and Shvaiko 2013; Arnold and Rahm 2014). Due to the typically high semantic heterogeneity of schemas, algorithms can determine approximate mappings only. The automatically determined mappings may thus require the inspection and adaptation by human domain experts (deletion of wrong correspondences, addition of missing correspondences) to obtain the correct mapping. Mapping results are useful input for further data integration tasks such as merging or integrating the respective schemas since a mapping indicates

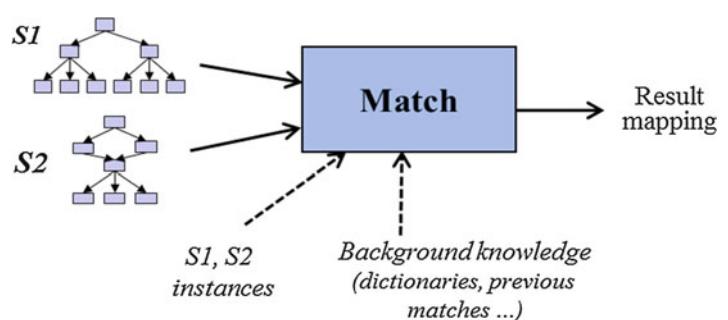
the elements that should be represented only once in the integrated result. In fact, several such mapping-based merge approaches have been proposed for both schemas and ontologies (Pottinger and Bernstein 2003; Raunich and Rahm 2014).

Schema matching has been a very active research area in the last decades, and numerous techniques and prototypes for automatic matching have been developed (Rahm and Bernstein 2001; Euzenat and Shvaiko 2013). Matches are usually identified by a combination of techniques (also called *matchers*) to determine the similarity of elements. These include

- Linguistic matchers that focus on the similarity of element names based on string similarity measures or synonym information from background knowledge resources such as dictionaries
- Structural matchers that determine the similarity of elements based on the similarity of their schema neighbors, e.g., ancestors and/or descendants
- Instance matchers that determine the similarity of elements based on their associated instance data, e.g., attribute values or entities of a concept

Despite the advances made, current match systems still struggle to deal with large-scale match tasks. In particular, achieving both good efficiency and good effectiveness are two major challenges. Further challenges for large-scale schema matching include benchmarking and evaluation as well as suitable approaches and interfaces for user involvement. The latter issues are further

Large-Scale Schema Matching, Fig. 1 Input and output of an automatic schema matching system



discussed in Hung et al. (2013) and are beyond the scope of this entry.

Key Research Findings

We focus on recent approaches to improve efficiency and effectiveness for large-scale pairwise schema (and ontology) matching.

The main performance (efficiency) problem for matching large schemas is the potentially huge search space if one has to compare every element of the first with every element of the second input schema. For schemas and ontologies with thousands or tens of thousands of elements, this leads to a search space of many million comparisons. This cannot only result in long execution times for the evaluation of several matchers but will also make it difficult to find the relatively few correspondences under so many possible candidates. Hence, it is imperative for large-scale matching to reduce the search space. Furthermore, one can apply parallel matching to improve runtimes. Note that these techniques (reduction of the search space and parallel matching) are also common for large-scale entity matching (Kolb and Rahm 2013; Dong and Srivastava 2015).

Parallel matching is relatively straightforward but has been applied in only few prototypes for schema matching, e.g., in Gomma (Gross et al. 2010) and SPHeRe (Amin et al. 2016). It can be used to speed-up single matchers, e.g., by partitioning the search space (set of element pairs from the input schemas) and evaluating different partitions in parallel on different processors. Furthermore, different matchers can be evaluated in parallel to each other.

Reducing the search space for matching large schemas has found more interest. One idea is to apply several matchers in sequence and to initially use fast matcher(s) (e.g., a string matcher on element names) with a low similarity threshold to already eliminate most element pairs before applying further matchers (Ehrig and Staab 2004; Peukert et al. 2010). Several other approaches are based on a partitioning of the input schemas and restrict the search of similar schema elements to

a subset of these partitions. Such *partition-based matching* can achieve a significant reduction of the search space and thus improved efficiency. Furthermore, matching the smaller partitions reduces the memory requirements compared to matching the full schemas. The partition-based match tasks can also be performed in parallel for additional performance improvements (Amin et al. 2016).

There are many possible ways to perform partitioning and finding the most effective approaches is still an open research problem. COMA++ was one of the first systems to support partition-based schema matching by a so-called fragment matching (Do and Rahm 2007). Fragment matching works in two phases. In the first phase, the fragments of a specified type (e.g., subschemas such as relational tables) are determined and compared with each other to identify the most similar fragments from the other schema worth to be fully matched later. The search for similar fragments is some kind of light-weight matching, e.g., based on the similarity of the fragment roots. In the second phase, each pair of similar fragments is matched independently to identify correspondences between their elements.

The ontology matching system Falcon-AO (Hu et al. 2008) uses a structural clustering to initially partition the ontologies into relatively small, disjoint blocks. Matching is then restricted to the most similar blocks from the two ontologies. To determine the block similarity, Falcon-AO utilizes so-called anchors, i.e., highly similar element pairs that are determined before partitioning by a combined name/comment matcher. Matching is then limited to pairs of blocks sharing at least one anchor. Further approaches for partition-based matching used different kinds of clustering for the input schemas/ontologies, e.g., hierarchical clustering in Albergawy et al. (2011).

The taxonomy matching system AnchorFlood implements a dynamic partition-based matching that avoids the a-priori partitioning of the ontologies (Hanif and Aono 2009). Like in Falcon-AO, the approach utilizes anchors (similar concept pairs) but takes them as a starting point to incrementally match elements in their structural

neighborhood until no further matches are found or all elements are processed. Thus, the partitions (called segments) are located around the anchors and their size depends on the continued success of finding match partners for the considered elements.

Several approaches considered the case when one of the input schemas/ontologies is much smaller than the other one so that only the larger input is partitioned (Zhong et al. 2009; Gross et al. 2012). In GOMMA (Gross et al. 2012), a fast name matcher is initially applied between the small and larger ontology. Matching is then limited to the subontologies from the larger ontology for which the number of correspondences exceeds a certain ratio of all correspondences.

Effectiveness (high match quality) is another key challenge that requires the correct and complete identification of semantic correspondences. This problem becomes more difficult for larger search spaces. Furthermore, the semantic heterogeneity is typically high for large-scale match tasks, e.g., the schemas may largely differ in their size and scope making it difficult to find all correspondences and to avoid false match candidates. In addition to the combined use of several matchers, the following techniques are especially promising to improve match effectiveness for large-scale schema matching (Rahm 2011):

- *Postprocessing and repair of proposed correspondences.* The individual matchers typically generate a large number of possible correspondences from which the final ones need to be selected before they are included in the match result, e.g., for possible validation by a domain expert. The selection step should not only consider different ways to aggregate matcher results (union, intersection, weighted similarity values, etc.) and similarity thresholds but should also consider global constraints, e.g., if there should be at most 1 or a small number k of match partners in the other schema. So it is reasonable for 1:1 mappings to enforce so-called stable marriages or to only accept the mutually best match candidates, i.e., a correspondence $c1-c1'$ is only

accepted if $c1'$ is the most similar element for $c1$ and vice versa (Max-Both selection in COMA).

For semantically expressive ontologies, e.g., specified in the language OWL-DL, the determined mapping M between two ontologies may introduce inconsistencies so that axioms in the input ontologies may be violated if one combines $O1$ and $O2$ with the correspondences of M within a merged ontology. For example, for relations “ $O1.Customer$ is-a $O1.Person$ ” and “ $O2.Person$ is-a $O2.Member$ ” the equality correspondences “ $O1.Person \approx O2.Person$ ” and “ $O1.Customer \approx O2.Member$ ” would result in an inconsistency because “ $O1.Person$ is-a $O1.Customer$ ” could then be derived. The task of *mapping repair* (Meilicke et al. 2007) is to eliminate correspondences during post-processing that would cause such inconsistencies (e.g., “ $O1.Customer \approx O2.Member$ ” in the example). Mapping repair is supported in some ontology matching systems, e.g., LogMap and AlComo. It has to be noted that mapping repair can reduce recall by eliminating correct correspondences. Furthermore, the input ontologies may follow partially conflicting way in the representation of a domain that cannot be resolved by automatic repair actions but only with manual interaction.

- *Utilization of background knowledge.* The information within the input schemas itself is insufficient for large-scale matching with heterogeneous inputs so that it becomes necessary to find correspondences with the help of general or domain-specific background knowledge in dictionaries and thesauri (e.g., Wordnet, UMLS).
- *Reuse of previous match results.* Already matched schemas and the match results can be maintained in a repository for later reuse, in particular when modified versions of the schemas or other similar schemas have to be matched. Reusing such already found correspondences is a special form of utilizing background knowledge that holds high promise but also requires a complex infrastructure to find the reusable

correspondences and schema mappings. The corpus-based match approach of (Madhavan et al. 2005) uses a domain-specific corpus of schemas and match mappings and focuses on the reuse of element correspondences. They augment schema elements with matching elements from the corpus and assume that two schema elements match if they match with the same corpus element(s).

COMA was the first match tool to reuse entire schema mappings and to support indirect matching by composing already existing mappings (Do and Rahm 2002). For example, a mapping between schemas S1 and S3 can be obtained by combining preexisting S1-S2 and S2-S3 mappings involving another schema S2. Such derived mappings can also be combined with the results of regular matchers. The reuse and composition approach of COMA also allows the adaptation of an old mapping S1-S2 after one of the schemas evolves, e.g., S2 to S2', by composing the previous match result with the newly determined mapping S2-S2' between the old and new version of S2. The reuse and composition approach has also been incorporated and extended within the GOMMA prototype for ontology matching (Gross et al. 2011).

- *Semi-automatic tuning of match strategies.* In most current systems, the match strategy needs to be manually specified and configured, in particular the choice of matchers and the method to combine matcher results as well as critical parameters such as the weights of different matchers and similarity thresholds. Obviously, these decisions have a significant impact on match effectiveness but are difficult to make even for expert users. Semi-automatic tuning approaches aim at reducing this problem either by using learning-based or rule-based approaches. While supervised machine learning methods are very successful for semi-automatically configuring entity matching strategies, they are less applicable for learning schema matching configurations due to the difficulty to provide a sufficiently high number of correct schema mappings for training.

Several match systems including RIMOM analyze the schemas to be matched and determine their linguistic and structural similarity. These similarity characteristics are then used to select matchers or to weight the influence of different matchers in the combination of matcher results. Peukert et al. (2012) proposed a rule-based approach to tune entire match workflows based on features of the input schemas and of intermediate matcher results. There are different kinds of rules representing expert knowledge, in particular to select the matchers based on schema features and to combine matcher results based on schema and mapping features.

Future Directions

Despite the huge amount of research on schema matching, current products for data exchange and data integration typically include only simple linguistic approaches for automatically finding correspondences but still rely to a large degree on manual match decisions (Rahm 2011). As a result, they are ill-suited for large-scale match problems pointing to the need of bringing more proposed match approaches into products and practical application.

There is still a need for more research. In particular, it would be worthwhile to comparatively evaluate proposed approaches for mapping repair, reusing previous match results, and for self-tuning match strategies. Based on the observations, we envision possibilities to improve previous approaches in these areas.

Further topics for future research include improved user interfaces for controlling schema matching processes and providing expert feedback as well as holistic schema matching for multiple (>2) schemas.

Cross-References

- ▶ [Holistic Schema Matching](#)
- ▶ [Large-Scale Entity Resolution](#)

References

- Algergawy A, Massmann S, Rahm E (2011) A clustering-based approach for large-scale ontology matching. In: Proceedings of the advances in databases and information systems conference, vol 6909. Springer LNCS
- Amin MB, Khan WA, Hussain S, Bui DM, Banos O, Kang BH, Lee S (2016) Evaluating large-scale biomedical ontology matching over parallel platforms. IETE Tech Rev 33(4):415–427
- Arnold P, Rahm E (2014) Enriching ontology mappings with semantic relations. Data Knowl Eng 93: 1–18
- Do HH, Rahm E (2002) COMA – A system for flexible combination of schema matching approaches. In: Proceedings of the very large data bases conference, pp 610–621
- Do HH, Rahm E (2007) Matching large schemas: approaches and evaluation. Inf Syst 32(6):857–885
- Dong XL, Srivastava D (2015) Big data integration. Morgan & Claypool, San Rafael
- Ehrig M, Staab S (2004) Quick ontology matching. In: Proceedings of the international conference semantic web, vol 3298. Springer LNCS
- Euzenat J, Shvaiko P (2013) Ontology matching, 2nd edn. Springer, Berlin/Heidelberg, Germany
- Gross A, Hartung M, Kirsten T, Rahm E (2010) On matching large life science ontologies in parallel. In: Proceedings of the data integration in the life sciences conference, vol 6254. Springer LNCS
- Gross A, Hartung M, Kirsten T and Rahm E (2011) Mapping composition for matching large life science ontologies. In: Proceedings of the international conference on biomedical ontology, pp 109–116
- Gross A, Hartung M, Kirsten T, Rahm E (2012) GOMMA results for OAEI 2012. In: Proceedings of the 7th ontology matching workshop. CEUR-WS 946
- Hanif MS, Aono M (2009) An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. J Web Semant 7(4): 344–356
- Hu W et al (2008) Matching large ontologies: a divide-and-conquer-approach. Data Knowl Eng 67(1): 140–160
- Hung NQV, Tam NT, Miklós Z, Aberer K (2013) On leveraging crowdsourcing techniques for schema matching networks. In: Proceedings of the database systems for advanced applications, vol 7826. LNCS Springer
- Kolb L, Rahm E (2013) Parallel entity resolution with Dedoop. Datenbank-spektrum 13:23. Springer
- Madhavan J, Bernstein PA, Doan A, Halevy AY (2005) Corpus-based Schema matching. In: Proceedings of the IEEE ICDE conference
- Meilicke C, Stuckenschmidt H, Tamlin A (2007) Repairing ontology mappings. In: Proceedings of the AAAI, pp 1408–1413
- Peukert E, Berthold H, Rahm E (2010) Rewrite techniques for performance optimization of schema matching processes. In: Proceedings of the extending database technology conference
- Peukert E, Eberius J, Rahm E (2012) A self-configuring schema matching system. In: Proceedings of the IEEE ICDE conference, pp 306–317
- Pottinger RA, Bernstein PA (2003) Merging models based on given correspondences. In: Proceedings of the very large data bases conference, pp 862–873
- Rahm E (2011) Towards large-scale schema and ontology matching. In: Schema matching and mapping. Springer, Heidelberg, Germany
- Rahm E (2016) The case for holistic data integration. In: Proceedings of the advances in databases and information systems conference, vol 9809. Springer LNCS, Berlin/Heidelberg
- Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350
- Raunich S, Rahm E (2014) Target-driven merging of taxonomies with ATOM. Inf Syst 42:1–14
- Zhong Q, Li H et al (2009) A Gauss function based approach for unbalanced ontology matching. In: Proceedings of the ACM SIGMOD conference

Lineage

- [Data Provenance for Big Data Security and Accountability](#)

Link Analysis

- [Link Analytics in Graphs](#)

Link Analytics in Graphs

Peixiang Zhao
Department of Computer Science, Florida State University, Tallahassee, FL, USA

Synonyms

[Link analysis](#); [Link mining](#)

Definitions

Link analytics is a set of specialized data analysis and graph mining techniques that discover, examine, and evaluate the *relationships* or *inter-linked* structures of graphs.

Overview

Graph-structured data are ubiquitous (Aggarwal and Wang 2010; Cook and Holder 2006), which consist of *vertices* (or *nodes*) representing physical, technological, conceptual, and societal entities or objects and *edges* (or *links*) illustrating connections, relationships, or dependencies between vertices in application-specific ways. Noteworthy examples of graphs and networked data include the World Wide Web, where webpages are vertices and hyperlinks are edges (Kleinberg et al. 1999), and social networks, where individuals are vertices and friendship relations are edges (Pitas 2015). In response to the growing popularity and wide applicability of graphs, a proliferation of link analysis techniques has emerged, focusing primarily on the modeling, quantification, mining, and evaluation of potentially useful, structure-enriched information from graphs, including, but not limited to, **link-based ranking** and **link prediction**.

Key Research Findings

Link-Based Ranking

Given a graph G , link-based ranking methods aim to rank vertices of G based upon their *structure significance* modeled and quantified based in particular on the edges of G . Specifically, each vertex v of G is associated with (relative) quantitative assessment values indicating the significance of v throughout G . Representative link-based ranking methods include PageRank (Brin and Page 1998), Personalized PageRank (Jeh and Widom 2003; Page et al. 1998), HITS (Kleinberg 1999), and SimRank (Jeh and Widom 2002).

PageRank

PageRank is a link-based ranking algorithm, the objective of which is to assign a numerical score, called *PageRank score*, to each vertex by exploiting interlinked structures of the graph G . The PageRank score of a vertex v can be regarded as a “vote”, by all the other vertices of G , about how important v is. Empirically, A link to v counts as a vote of support for v . A vertex with a high PageRank score is usually considered more “important” or more “influential” than a vertex with a low PageRank score. In principle, to compute the PageRank score of a vertex v , denoted as $P(v)$, we consider all the vertices u that links to v , i.e., for each such u , there exists an edge (u, v) in the graph G . If the degree of u , $\deg(u)$, is n , then u contributes $\frac{1}{\deg(u)}$ of its PageRank score to that of v :

$$P(v) = \sum_{u:(u,v) \in G} \frac{P(u)}{\deg(u)} \quad (1)$$

To account for the vertices with no outbound links, their PageRank scores can be regarded as being divided evenly among all the other vertices of G . As a result, a parametric damping factor d is introduced, and the PageRank formulation is refined as

$$P(v) = \frac{1-d}{N} + d \sum_{u:(u,v) \in G} \frac{P(u)}{\deg(u)} \quad (2)$$

where N is the number of vertices of G .

Based on the random surfer model (Chebolu and Melsted 2008), PageRank measures the stationary distribution of one specific kind of random walk that starts from a random vertex u of G ; and in each iteration, with a predefined probability $p = 1 - d$, jumps to a random vertex; and, with probability $1 - p = d$, follows a random outgoing edge of the current vertex u . PageRank scores can be approximated by the Power method with a high degree of accuracy (Bahmani et al. 2010; Berkhin 2005). It has been reported that the PageRank algorithm, once ran upon a graph of 322 million edges, converged within a tolerable limit in just 52 iterations (Brin and Page 1998).

Personalized PageRank

Personalized PageRank is the personalized, ego-centric version of the PageRank algorithm (Lofgren et al. 2014). Given a graph G and a starting vertex v , Personalized PageRank assigns a score to every other vertex u of G . This score models how much v is interested in u or how much v trusts u . From the perspective of random Markov theory, Personalized PageRank is almost the same as PageRank, except that all the random jumps, with a predefined probability p , are made to the starting vertex v for which we are personalizing the PageRank, as opposed to any vertex of the graph G .

The exact personalized PageRank scores for all vertices of G with respect to a particular source vertex v can be computed by the Power method (Maehara et al. 2014; Page et al. 1998), which involves costly matrix multiplication operations. Furthermore, materializing the personalized PageRank score for each vertex v of G is clearly infeasible for large graphs. As a result, most existing methods have focused on approximate personalized PageRank computation for large personalized PageRank scores with accuracy guarantees (Wang et al. 2016; Fujiwara et al. 2013).

HITS

The Hyperlink-Induced Topic Search (HITS) algorithm is a link analysis method that was initially designed for webpage search. Given a user-specified query, a subgraph G of all relevant vertices to the query is first selected from the original graph. For each vertex v of G , two scores are further assigned to indicate the importance of v : *authority* and *hub*. Intuitively, v is considered an authority if it provides direct answers to specific information needs, and there exist many hub vertices linking to it. Likewise, v is considered a hub if it provides a good list of links to the high-quality vertices; that is, v points to many other authoritative vertices. As a result, authority and hub values are defined in terms of one another in a mutually recursive fashion: the authority of v is computed as the sum of the hub values for the vertices that point to v :

$$\text{authority}(v) = \sum_{u:(u,v) \in G} \text{hub}(u) \quad (3)$$

and the hub value of v is the sum of the authority values for the vertices to which v points:

$$\text{hub}(v) = \sum_{u:(v,u) \in G} \text{authority}(u) \quad (4)$$

The HITS algorithm iterates by updating the two scores for each vertex of G . In order to ensure convergence of the HIT algorithm, both the authority and hub scores are scaled and normalized within each iteration. In practice, after a number of iterations when both the authority and hub scores of vertices do not vary significantly, the HITS algorithm can be considered to have converged (Kleinberg 1999).

Consider the adjacency matrix A of the graph G , which is an $N \times N$ symmetric matrix, where N is the number of vertices of G . The matrix entry A_{ij} ($1 \leq i, j \leq N$) is 1 if there exists an edge from vertex v_i to v_j and 0 otherwise. We further denote the hub vector $\mathbf{hub} = \{\text{hub}(v_1), \dots, \text{hub}(v_N)\}$, where $\text{hub}(v_i)$ ($1 \leq i \leq N$) is the hub score of the vertex v_i . Likewise we denote the authority vector $\mathbf{authority} = \{\text{authority}(v_1), \dots, \text{authority}(v_N)\}$. The HITS algorithm in matrix notations can therefore be formulated as

1. computer $\mathbf{hub} = A \times \mathbf{authority}$;
2. computer $\mathbf{authority} = A^T \times \mathbf{hub}$;
3. Iterate until convergence.

By substitution, it is immediate that

$$\mathbf{hub} = A \times A^T \times \mathbf{hub} \quad (5)$$

and

$$\mathbf{authority} = A^T \times A \times \mathbf{authority} \quad (6)$$

That is, \mathbf{hub} is an eigenvector of AA^T and $\mathbf{authority}$ is an eigenvector of A^TA . As a result, the HITS algorithm is actually a special case of the power method, and both

HITS and PageRank algorithms formalize link-based ranking into the eigenvector problems of designated matrixes (Ding et al. 2002).

SimRank

Given a graph G , it is important to assess the similarity of vertices based upon the pure, interlinked graph topology. Among a number of similarity measures, SimRank has been recognized as one of the most well adopted (Jiang et al. 2017; Tian and Xiao 2016). SimRank is proposed based on the following intuitive argument: “*two vertices are considered similar if they are referenced by similar vertices.*” As the base case, we consider a vertex maximally similar to itself, to which we can assign the SimRank score of 1. Furthermore, the similarity between two different vertices u and v is defined as:

$$s(u, v) = \frac{c}{|I(u)| \cdot |I(v)|} \sum_{a \in I(u), b \in I(v)} s(a, b) \quad (7)$$

where $I(v)$ denotes the set of neighboring vertices pointing to v and $c \in (0, 1)$ is a decay factor. A solution to the SimRank equation can be reached by an iterative computation to a fixed-point (Lizorkin et al. 2008).

SimRank can also be interpreted in terms of coupled random walks. Consider any two vertices u and v of G , and we start random walks from u and v , respectively, such that the two random walks are always with the same length t and they meet, for the first time, at a vertex w of G . Then the SimRank score $s(u, v)$ is equivalent to the expected f -meeting distance (Jeh and Widom 2002):

$$s(u, v) = \sum_{t=0}^{\infty} c^t \times \sum_{w \in G} \Pr(u, v, w) \quad (8)$$

where $\Pr(u, v, w)$ is the probability of two random walks originated from u and v , respectively, that meet at w for the first time.

Link Prediction

Real-world graphs are not static but dynamically evolving with new vertices and edges added all the time. It is therefore fundamental and critical to understand the dynamics and evolution of graphs. In particular, consider two vertices u and v of a graph G , where u and v are not connected via an edge. An interesting problem that has fused intensive research interest and found widely varying applications is to predict, given the current state of the graph G , the likelihood of a future connection between u and v , that is, a new edge (u, v) (Liben-Nowell and Kleinberg 2003). This problem is commonly referred to as the *link prediction* problem (Martínez et al. 2016; Duan et al. 2016; Barbieri et al. 2014; Hasan and Zaki 2011).

We can model the link prediction problem as a supervised classification task, where the current graph snapshot is used as the training data to build the link prediction model and the predictions of future links can be made afterward. This is a typical binary classification task, in that our main goal is to tell, given two nonadjacent vertices u and v , if or not there will be an edge (u, v) in the near future. As a result, the existing supervised classification methods, including naive Bayes, neural networks, support vector machines (SVM), and k -nearest neighbors, can be used. The key challenge here is to select a set of appropriate, link-based features for classification.

The link-based features that are leveraged for link prediction are primarily based on the pure graph topology. Typically, they define vertex-wise similarity based on linked structures, or the ensembles of paths, between vertices (Liben-Nowell and Kleinberg 2003). The most well-known link-based features are discussed below.

- **Common Neighbors.** Given two vertices u and v , the size of their common neighboring vertices in G is defined as $|N(u) \cap N(v)|$, where $N(\cdot)$ denotes the set of adjacent vertices for a given vertex. Intuitively speaking, if u and v share a lot of common neighboring vertices (e.g., friends), then u and v will connect with each other with high probability in the future;

- **Jaccard Coefficient.** The Jaccard coefficient metric normalizes common neighbors as follows,

$$\text{Jaccard}(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \quad (9)$$

Equivalently, Jaccard coefficient can be interpreted as the probability of selecting a common neighbor of u and v from the union of the neighbors of u and v ;

- **Preferential Attachment** In social networks, the users who have had many friends already tend to connect more in the future, which is commonly referred to as the *rich-get-richer* phenomenon. We can quantify “richness” of a vertex by its degree, and the preferential attachment measure is thus defined as

$$\text{P-A}(u, v) = |N(u)| \cdot |N(v)| \quad (10)$$

Note that preferential attachment does not require any detailed information for vertex neighbors; therefore, it has the lowest computational complexity;

- **Adamic-Adar.** The Adamic-Adar weighs the common neighbors of u and v with smaller degrees more heavily,

$$\text{Adamic}(u, v) = \sum_{w \in N(u) \cap N(v)} \frac{1}{\log|N(w)|} \quad (11)$$

Intuitively, if both u and v share a common neighbor w , which turns out to be a high-degree vertex connecting to many other vertices as well, the effect of influencing the future connection between u and v , in terms of w , should be dampened;

- **Shortest Path Distance.** Empirically speaking, the shorter the distance between u and v in G , the higher the probability u and v will be directly connected in the future. Due to the small-world phenomenon (Watts and Strogatz 1998), however, most vertex pairs in real-world graphs are separated by fairly short distances. As a result, this path-based feature sometimes leads to poor link prediction performance;

- **Katz.** The Katz measure can be viewed as a variant of shortest path distance:

$$\text{Katz}(u, v) = \sum_{l=1}^{\infty} \beta^l \cdot |\text{paths}_{u,v}^l| \quad (12)$$

where $\beta (\leq 1)$ is a dampening parameter to penalize the paths with long path lengths, l denotes the length of a path, and $\text{paths}_{u,v}^l$ is the set of all the paths of length l between u and v in G . Katz considers the ensemble of all paths between u and v and thus generally works much better than shortest path distance in link prediction. However, computing Katz in real-world graphs turns out to be very expensive;

- **Hitting Time.** Given two vertices u and v in a graph G , the hitting time, $H(u, v)$, is the expected number of steps of a random walk starting at u to reach v . A shorter hitting time means that u and v are more similar, and thus there exists a higher probability that u and v will be linked together in the future. It is easy to compute $H(u, v)$ by initiating a sample of random walks. However, its value may have high variances. As a result, link prediction based on hitting time may result in poor prediction performance;

- **Rooted PageRank.** The hitting time measure is sensitive to the vertices that are far away from u and v to be examined, even if u and v are very close to each other in G . To alleviate this problem, we allow the random walk from u to v to periodically restart back to u with a fixed probability α at each random step. This way, distant part of G that is far away from u and v will almost never be explored. This approach results in the Rooted PageRank measure, which is reminiscent of Personalized PageRank in the link-based ranking methods.

Besides the aforementioned link feature-based methods, there are also Bayesian probabilistic methods, probabilistic relational methods, and linear algebraic methods for link prediction. Readers are referred to the link prediction surveys for more technical details (Hasan and Zaki 2011).

Examples of Applications

Link analytics involves data analysis techniques used in network science to evaluate the relationships in graphs. It is essentially a kind of knowledge discovery that can be used in widely varying real-world, graph-structured applications, including search engine optimization, security analysis, and medical research.

The analysis of hyperlinks and the graph structure of the Web has been instrumental in the development of web search. The use of hyperlinks for ranking web search results is probably one of the most noteworthy examples (Brin and Page 1998; Page et al. 1998). PageRank and HITS algorithms have been the key factors considered by web search engines in computing a composite score for a web page on any given query. Over the last decade, both of them have emerged as a very effective measure of reputation for web graphs and social networks (Liu et al. 2017).

Giving higher weights to the nearby vertices in Personalized PageRank has enabled it to find many applications in different real-world graphs, including friend recommendation in social networks (Gupta et al. 2013; Backstrom and Leskovec 2011) and graph partitioning (Andersen et al. 2006). Personalized PageRank has also been used to rank items in bipartite graphs for recommendation (Bahmani et al. 2010). Furthermore, it has also been applied from biology to chemistry and to civil engineering (Fogaras et al. 2005).

SimRank is a general link-based similarity measure, which is computed solely based on the vertices' structural context. SimRank has been successfully employed in many graph-based applications, such as sponsored search and web spam detection (Antonellis et al. 2008) and schema matching (Melnik et al. 2002).

Link prediction has found a series of applications in social networks. For instance, in numerous social media websites, with the “friend recommendation” feature, they will suggest users connected with other potential friends in real life, or they may suggest friends you already know but just not yet connected. Beyond social network applications, link prediction has been used to find interactions between proteins (Airoldi et al.

2008). In the security domain, link prediction can help identify hidden groups of terrorists or criminals (Hasan and Zaki 2011).

Future Directions for Research

Link analytics has been a promising research direction in data science and graph mining and thus generated a series of disruptive and influential techniques and methodologies that have significantly shaped the modern networked world. There exist quite a few research frontiers for link analytics that are worthy of systematic and thorough studies in the future:

1. Real-world graphs and networks are often-times enormous in sizes and scales, making existing solutions for link analytics hard to adapt to the so-called big graphs. New link analysis principles and methodologies that can scale up or scale out upon Internet-scale graphs will be of great importance in the Big Data Era;
2. Real-world graphs are typically generated from disparate, heterogeneous data sources, thus resulting in heterogeneous, multidimensional graph data. The synergy and unification of graph topology and heterogeneous contents will enhance both the effectiveness and efficiency for existing link analysis methods;
3. Real-world graphs are not static but dynamically evolving in fast rates and speed. Enabling real-time and accurate link analytics upon fast, dynamically evolving graphs will further spur extensive research interest and potential applications for dynamic graphs and graph streams.

L

Cross-References

- ▶ [Graph Data Models](#)
- ▶ [Graph Representations and Storage](#)
- ▶ [Linked Data Management](#)

References

- Aggarwal CC, Wang H (2010) Managing and mining graph data, 1st edn. Springer Publishing Company, Inc., Boston

- Airoldi EM, Blei DM, Fienberg SE, Xing EP (2008) Mixed membership stochastic blockmodels. *J Mach Learn Res* 9:1981–2014
- Andersen R, Chung F, Lang K (2006) Local graph partitioning using pagerank vectors. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS'06), pp 475–486
- Antonellis I, Molina HG, Chang CC (2008) Simrank++: query rewriting through link analysis of the click graph. *Proc VLDB Endow* 1(1):408–421
- Backstrom L, Leskovec J (2011) Supervised random walks: predicting and recommending links in social networks. In: Proceedings of the fourth ACM international conference on web search and data mining (WSDM'11), pp 635–644
- Bahmani B, Chowdhury A, Goel A (2010) Fast incremental and personalized pagerank. *Proc VLDB Endow* 4(3):173–184
- Barbieri N, Bonchi F, Manco G (2014) Who to follow and why: link prediction with explanations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'14), pp 1266–1275
- Berkhin P (2005) Survey: a survey on pagerank computing. *Internet Math* 2(1):73–120
- Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the seventh international conference on World Wide Web (WWW'98), pp 107–117
- Chebolu P, Melsted P (2008) Pagerank and the random surfer model. In: Proceedings of the nineteenth annual ACM-SIAM symposium on discrete algorithms (SODA'08)
- Cook DJ, Holder LB (2006) Mining graph data. Wiley, New York
- Ding C, He X, Husbands P, Zha H, Simon HD (2002) Pagerank, hits and a unified framework for link analysis. In: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval (SIGIR'02), pp 353–354
- Duan L, Aggarwal C, Ma S, Hu R, Huai J (2016) Scaling up link prediction with ensembles. In: Proceedings of the ninth ACM international conference on web search and data mining (WSDM'16), pp 367–376
- Fogaras D, Rácz B, Csalogány K, Sarlós T (2005) Towards scaling fully personalized pagerank: algorithms, lower bounds, and experiments. *Internet Math* 2(3):333–358
- Fujiwara Y, Nakatsui M, Shiokawa H, Mishima T, Onizuka M (2013) Efficient ad-hoc search for personalized pagerank. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data (SIGMOD'13), pp 445–456
- Gupta P, Goel A, Lin J, Sharma A, Wang D, Zadeh R (2013) WTF: The who to follow service at twitter. In: Proceedings of the 22nd international conference on World Wide Web (WWW'13), pp 505–514
- Hasan M, Zaki M (2011) A survey of link prediction in social networks. In: Aggarwal CC (ed) Social network data analytics. Springer, New York, pp 243–275
- Jeh G, Widom J (2002) Simrank: a measure of structural-context similarity. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'02), pp 538–543
- Jeh G, Widom J (2003) Scaling personalized web search. In: Proceedings of the 12th international conference on World Wide Web (WWW'03), pp 271–279
- Jiang M, Fu AWC, Wong RCW (2017) Reads: a random walk approach for efficient and accurate dynamic simrank. *Proc VLDB Endow* 10(9):937–948
- Kleinberg JM (1999) Authoritative sources in a hyper-linked environment. *J ACM* 46(5):604–632
- Kleinberg JM, Kumar R, Raghavan P, Rajagopalan S, Tomkins AS (1999) The web as a graph: measurements, models, and methods. In: Proceedings of the 5th annual international conference on computing and combinatorics (COCOON'99), pp 1–17
- Liben-Nowell D, Kleinberg J (2003) The link prediction problem for social networks. In: Proceedings of the twelfth international conference on information and knowledge management (CIKM'03), pp 556–559
- Liu Q, Xiang B, Yuan NJ, Chen E, Xiong H, Zheng Y, Yang Y (2017) An influence propagation view of pagerank. *ACM Trans Knowl Discov Data* 11(3):30:1–30:30
- Lizorkin D, Velikhov P, Grinev M, Turdakov D (2008) Accuracy estimate and optimization techniques for simrank computation. *Proc VLDB Endow* 1(1):422–433
- Lofgren PA, Banerjee S, Goel A, Seshadhri C (2014) FAST-PPR: scaling personalized pagerank estimation for large graphs. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'14), pp 1436–1445
- Maehara T, Akiba T, Iwata Y, Kawarabayashi Ki (2014) Computing personalized pagerank quickly by exploiting graph structures. *Proc VLDB Endow* 7(12):1023–1034
- Martínez V, Berzal F, Cubero JC (2016) A survey of link prediction in complex networks. *ACM Comput Surv* 49(4):69:1–69:33
- Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: Proceedings of the 18th international conference on data engineering (ICDE'02), pp 117–128
- Page L, Brin S, Motwani R, Winograd T (1998) The pagerank citation ranking: bringing order to the web. In: Proceedings of the 7th international World Wide Web conference (WWW'98), pp 161–172
- Pitas I (2015) Graph-based social media analysis. Chapman & Hall/CRC, Boca Raton
- Tian B, Xiao X (2016) Sling: A near-optimal index structure for simrank. In: Proceedings of the 2016 international conference on management of data (SIGMOD'16), pp 1859–1874
- Wang S, Tang Y, Xiao X, Yang Y, Li Z (2016) Hubppr: effective indexing for approximate personalized pagerank. *Proc VLDB Endow* 10(3):205–216

Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. *Nature* 393(6684):440–442

Link Discovery

► Large-Scale Entity Resolution

Link Mining

► Link Analytics in Graphs

Linked Data

► Data Quality and Data Cleansing of Semantic Data

Linked Data Compression

► RDF Compression

Linked Data Management

Olaf Hartig¹, Katja Hose², and Juan Sequeda³

¹Linköping University, Linköping, Sweden

²Aalborg University, Aalborg, Denmark

³Capsenta, Austin, TX, USA

Synonyms

Linked data query processing; Web of data

Definitions

The term *Linked Data* refers to data that is made available on the Web in a structured, directly

machine-processable form via the following four principles:

1. Use URIs as names for things
2. Use HTTP URIs so that people [and machines] can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things. (Berners-Lee 2006; Bizer et al. 2009)

Per these principles, given a URI that names something (e.g., a person, a book, a concept, an abstract idea) [Principle 1], it is possible to look up this URI using the HTTP protocol [Principle 2] and retrieve a document with RDF data describing the thing [Principle 3]. This data may include other URIs to refer to other things, and these URIs can be looked up in the same manner [Principle 4].

L

Overview

When applying the abovementioned principles, data becomes interlinked on the Web. These links may not only point to additional data from the same website but also to other websites. This results in the emergence of a decentralized, globally distributed Web of Linked Data.

Since their introduction, these Linked Data principles have been adopted by various publishers (Bizer et al. 2009). Today, there are significant amounts of Linked Data accessible on the Web (Mika and Potter 2012; Schmachtenberg et al. 2014).

In addition to publishing data on the Web via a Linked Data interface, many data publishers also support other RDF-based mechanisms to access their datasets. One approach is in the form of *RDF dumps* that provide the complete dataset as a single downloadable file serialized in an RDF format. Another approach is through *SPARQL endpoints*, which are Web services that can be called to obtain the result of executing any given SPARQL query over the dataset (Feigenbaum et al. 2013). Alternative query interfaces have become available that are deliberately limited in

the expressiveness of the queries they support; the purpose of these limitations is to achieve a restricted, more predictable server load per request and, thus, to increase the overall availability of the query-based interfaces. The most prevalent of these alternative query interfaces is the *Triple Pattern Fragment (TPF)* interface (Verborgh et al. 2016).

While all these data publishing approaches are based on the RDF data model, it has to be emphasized that an isolated set of RDF triples (e.g., in an RDF dump or behind a SPARQL endpoint) is not Linked Data per se; what would make it Linked Data is the possibility to discover it on the Web by the aforementioned principles. Hence the fourth principle of including links between different RDF datasets is vital.

The remainder of this article first provides a general overview of different data management options to make use of Linked Data in applications and, thereafter, focuses on approaches to query Linked Data live on the Web by relying solely on the Linked Data publishing principles (i.e., URI lookups).

are a number of commercial triple stores such as AllegroGraph, Amazon Neptune, Blazegraph, GraphDB, MarkLogic, Stardog, and Virtuoso. For an overview of research on techniques to build highly scalable triple stores, refer to other entries in this encyclopedia (see the cross-references section).

In contrast to the following options, copying data into a dedicated triple store setup for the sole purpose of supporting a particular application is perhaps the most efficient option in terms of query performance. A possible downside is that the triple store has to be maintained and the copied data may have to be kept in sync with the data in the original data sources on the Web.

Distributed Client-Server Processing. If the application uses a single dataset only (or multiple in isolation) and the maintainer of this dataset provides a Web interface to query the dataset, the application can be built based on queries to be executed via this interface.

Such an interface may be a *SPARQL endpoint* (see above). However, Buil-Aranda et al. (2013) have shown empirically that public SPARQL endpoints on the Web may have availability issues.

The aforementioned *Triple Pattern Fragment interface* is a more reliable option (Verborgh et al. 2016). In this case however, to execute SPARQL queries over the dataset exposed via this TPF interface, the application requires a client-side query execution engine that is capable of processing such queries based on TPF requests. A number of TPF-based query execution approaches have been proposed (Verborgh et al. 2016; Van Herwegen et al. 2015) with the state of the art being an adaptive approach by Acosta and Vidal (2015) that adopts the idea of using Eddy operators (Avnur and Hellerstein 2000).

In addition to using pure TPF interfaces, some authors propose extended versions of this interface with the goal of reducing the shortcomings of TPF (e.g., significant network load) by retaining (or even increasing) the overall query throughput of TPF-based client-server systems. Examples of such proposals are to augment *TPF responses with membership metadata* such as

Key Research Findings

Various approaches toward Linked Data management and querying have been proposed in the literature. To provide an overview, we have categorized them into general approaches, foundations, source selection, source ranking strategies, query execution, and query optimization.

General Approaches to Make Use of Linked Data

Centralization. A first option to use Linked Data in an application is to obtain the application-relevant data (e.g., by crawling or by downloading the respective RDF dumps) and load this data into a database management system (DBMS) for RDF data that is set up for the application. DBMSs for RDF data, often referred to as *triple stores*, enable applications to query the data using SPARQL, the RDF query language. There

Bloom filters (Vander Sande et al. 2015) and the *bindings-restricted Triple Pattern Fragment* (*brTPF*) interface that allows client-side query engines to use variations of the semijoin algorithm (Hartig and Buil-Aranda 2016).

Federated Processing. Federated query processing engines go beyond the aforementioned client-server setting by supporting the execution of queries over a federation of multiple servers. Each of these servers typically provides access to a different dataset. Therefore, given a SPARQL query, the goal of a federated query execution is to return a query result that is the same as if the query was executed directly over the union of the datasets in the federation.

Existing approaches to federated query processing in the Linked Data context can be categorized into three classes.

The first class consists of approaches that assume Linked Data can be accessed via query-based Web interfaces (e.g., SPARQL endpoints, TPF). These approaches are discussed in more detail in the “► [Federated RDF Query Processing](#)” entry in this encyclopedia.

Approaches in the second class rely only on the Linked Data publishing principles (i.e., URI lookups) and, thus, do not require the additional availability of query-based interfaces to access the Linked Data. Such approaches, called *Linked Data query processing*, this article and will be discussed in the next sections.

The third class comprises *hybrid query approaches* which aim to leverage query-based interfaces to access Linked Data in combination with Linked Data query processing techniques. To date, the only proposals in this direction are by Umbrich et al. (2012) which combines Linked Data query processing with access to a SPARQL endpoint that serves as a query-able cache of Linked Data. Additionally, Lynden et al. (2013) propose to integrate data retrieval from a SPARQL endpoint into a Linked Data query execution process.

Foundations of Linked Data Queries

A *Linked Data query* is a query over a Web of documents of RDF data that can be retrieved by

URI lookups as per the Linked Data publishing principles. Such queries may be not only about the topology of the interlinked RDF documents but also, and perhaps more interesting, about the data in and across these documents. In particular, existing approaches to process Linked Data queries (as discussed in the next sections) focus on queries that are expressed using the conjunctive fragment of the SPARQL query language and that are to be evaluated in terms of the union of the data in the interlinked documents. An initial approach to providing a formal semantics for these queries has been proposed by Bouquet et al. (2009). More recently, Hartig (2012) has improved this work in three ways: (i) covering the whole core fragment of SPARQL (i.e., OPTIONAL, FILTER, UNION), (ii) showing formally that completeness of query results w.r.t. all Linked Data cannot be guaranteed, and (iii) introducing a more restrictive formal query semantics for which completeness can be guaranteed.

As an alternative to adopting SPARQL expressions as a language for Linked Data queries, a number of dedicated Linked Data query languages have been proposed, namely, LD-Path (Schaffert et al. 2012), NautiLOD (Fionda et al. 2015), and LDQL (Hartig and Pérez 2016). However, with the exception of the swglet approach (Fionda et al. 2014) for NautiLOD, techniques for processing Linked Data queries have focused on (conjunctive) SPARQL expressions.

Source Selection Strategies for Linked Data Queries

Any execution of Linked Data queries involves the retrieval of data by looking up URIs. Existing approaches for selecting the URIs to be looked up during the execution of a given query can be classified into the following three classes:

Index-based approaches determine the URIs to look up during query execution by using a pre-populated index (Harth et al. 2010; Umbrich et al. 2011). A typical example of such an index uses patterns of RDF triples as index keys (Ladwig and Tran 2010). Given such a pattern, the corresponding index entry is a set of URIs where

each of these URIs, when looked up, allows the query engine to retrieve some data that contains an RDF triple that matches the pattern. Further index structures for selecting URIs in Linked Data query processing have been studied by Harth et al. (2010), Umbrich et al. (2011), Tian et al. (2011), and Paret et al. (2011).

After such an index has been populated and can be used for query processing, the index has to be maintained. Maintenance may comprise indexing additionally discovered URIs and ensuring that the index is up to date (Umbrich et al. 2011). The latter is necessary because the data to be retrieved from indexed URIs may change over time. While such an index maintenance is similar to maintaining materialized views in a data warehouse, no work exists that explicitly studies approaches to maintain source selection indexes for Linked Data queries.

Live exploration approaches leverage the characteristics of Linked Data, in particular, the existence of data links. That is, to execute a given Linked Data query, live exploration systems perform a URI lookup process during which they incrementally discover further URIs that may be scheduled for lookup (Hartig et al. 2009; Hartig 2011b; Ladwig and Tran 2011; Schmedding 2011; Miranker et al. 2012). The data retrieved during this process is used not only to discover more URIs, but it also provides the basis for constructing the query result. Assuming the queries are expressed using SPARQL, live exploration approaches support naturally the aforementioned restrictive semantics for such queries as introduced by Hartig (2012), which also gives a well-defined boundary for the URI lookup process.

In comparison to index-based source selection approaches, live exploration approaches have the interesting characteristic of being able to discover initially unknown data sources *at query execution time*. Additionally, live exploration systems can readily be used without having to wait for the completion of an initial data load phase, index creation, or any other type of preprocessing.

However, given that the general ideas of index-based source selection and of live exploration

may be implemented in various ways, respectively, comparing both types of approaches in a fair manner is a nontrivial open problem.

Hybrid source selection approaches combine the two aforementioned ideas. So far, only one hybrid approach has been proposed in the literature. This approach uses an index to create a (ranked) list of URIs to look up; additional URIs discovered during the query execution process are then added into this list (Ladwig and Tran 2010).

Source Ranking Strategies for Linked Data Queries

In addition to selecting URIs that are to be looked up for a given Linked Data query, the selected URIs may be ranked in terms of the order in which these URIs have to be looked up. The goal of such a data source ranking may be to maximize the subset of the query result that can be found in a given amount of time.

Harth et al. (2010) and Ladwig and Tran (2010) have proposed data source ranking approaches for index-based source selection. These approaches rely on additional information about the data sources; the assumption is that this information is available in the index.

For selecting URIs using a live exploration approach, Hartig and Özsü (2016) have studied various heuristics to prioritize URI lookups; none of these heuristics uses any *a priori* information.

Linked Data Query Execution

Existing approaches to execute Linked Data queries as proposed in the literature can be classified into two categories:

Two-phase execution approaches separate the query execution process into two phases: During the first phase, all required data is retrieved from the Web (by looking up URIs, which may be selected using any of the aforementioned source selection approaches). Then, during the second phase, the query result is produced from the data retrieved in the first phase. Harth et al. (2010), Umbrich et al. (2011), and Paret et al. (2011) have adopted such a two-phase execution pro-

cess to study their index-based source selection approaches. While such a two-phase process is straightforward to implement, having to wait for the completion of the data retrieval phase may not be practical; it may even exceed the resources of the query execution system.

Integrated execution approaches integrate the retrieval of data with the result construction process. This allows a query engine to return query results incrementally, i.e., before data retrieval has been completed. As for the two-phase execution approaches, it is possible to use any type of source selection strategy as a basis for an integrated execution approach. Integrated execution approaches that use some form of live exploration are also referred to as *traversal-based query execution* approaches. A number of such approaches have been proposed where each of them is based on different implementation techniques: The first approach made use of the well-known iterator model (Hartig et al. 2009; Hartig 2011b, 2013). In contrast to such a pull-based implementation, Ladwig and Tran (2010, 2011) proposed a push-based implementation using the symmetric hash join operator. Other push-based implementations adapt the Rete match algorithm (Miranker et al. 2012) and the idea of an Eddy-style tuple routing operator (Hartig and Özsü 2016).

Query Optimization for Linked Data Queries

Besides the aforementioned source ranking techniques, which can be understood as a form of query optimization, the topic of query optimization for Linked Data queries is largely unexplored. In fact, there does not yet exist any work that investigates cost-based query optimization techniques in this context.

For traversal-based query execution, cost-based query optimization is particularly challenging because information to assess query plans may only be obtained incrementally during query execution. Hence, selecting an initial plan requires heuristics. Hartig (2011b) has proposed such plan selection heuristics for the aforementioned, iterator-based approach to

implement traversal-based query execution. For the same implementation approach, Hartig et al. (2009) introduce an extension of the iterator model to support a form of adaptive query optimization. This extension avoids a blocking behavior of iterators that may otherwise happen when waiting for the completion of particular URI lookups.

Future Directions for Research

Query optimization. As mentioned in the previous section, query optimization in the context of Linked Data query processing is a largely unexplored area. Possible topics for contributions are cost-based query optimization, multi-objective query optimization (i.e., taking into account not only the overall execution time but also, e.g., the time until the first output when returning query results incrementally or network traffic), techniques for adaptive query processing, and optimization for subsequent queries (e.g., collecting statistics as a side-product of traversal-based query executions and leveraging these statistics to optimize future queries).

Experimental comparisons. While a number of approaches for the different aspects of Linked Data query processing have been proposed, there exists little research that compares these approaches in a comprehensive and fair manner.

Benchmarks. Related to the previous point, there are no well-defined and well-understood benchmarks to test Linked Data query processing systems (neither for specific aspects of Linked Data query processing such as source selection, nor for the process of Linked Data query execution in general).

Techniques for more expressive query languages. All existing techniques for Linked Data query processing focus on queries expressed using the conjunctive fragment of SPARQL. However, the foundations of using more expressive fragments of SPARQL are well understood (Har-

tig 2012), and there even exist more fitting languages that have been specifically designed for Linked Data queries (Fionda et al. 2015; Hartig and Pérez 2016). It is an open question what are effective techniques for executing queries in these languages.

Hybrid query approaches. While there exists initial work on combining Linked Data query processing with other query paradigms such as query federation or queries over a centralized collection of data (Ladwig and Tran 2011; Umbrich et al. 2012; Hartig 2011a; Lynden et al. 2013), more research on such combinations is necessary. In particular, contributions can be made in terms of both establishing and understanding the foundations of such hybrid approaches as well as techniques for implementing hybrid approaches.

Cross-References

- ▶ [Federated RDF Query Processing](#)
- ▶ [Framework-Based Scale-Out RDF Systems](#)
- ▶ [Graph Data Management Systems](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Query Languages](#)
- ▶ [Native Distributed RDF Systems](#)
- ▶ [Semantic Interlinking](#)

References

- Acosta M, Vidal M (2015) Networks of linked data eddies: an adaptive web query processing engine for RDF data. In: The semantic web – ISWC 2015 – 14th international semantic web conference, Proceedings, Part I, Bethlehem, 11–15 Oct 2015, pp 111–127. https://doi.org/10.1007/978-3-319-25007-6_7
- Avnur R, Hellerstein JM (2000) Eddies: continuously adaptive query processing. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data, Dallas, 16–18 May 2000, pp 261–272. <http://doi.acm.org/10.1145/342009.335420>
- Berners-Lee T (2006) Design issues: linked data. Online at <http://www.w3.org/DesignIssues/LinkedData.html>
- Bizer C, Heath T, Berners-Lee T (2009) Linked data – the story so far. Int J Semantic Web Inf Syst (IJSWIS) 5(3):1–22
- Bouquet P, Ghidini C, Serafini L (2009) Querying the web of data: a formal approach. In: The semantic web, fourth Asian conference, ASWC 2009, Proceedings, Shanghai, 6–9 Dec 2009, pp 291–305. https://doi.org/10.1007/978-3-642-10871-6_20
- Buil-Aranda C, Hogan A, Umbrich J, Vandenbussche P (2013) SPARQL web-querying infrastructure: ready for action? In: The semantic web – ISWC 2013 – 12th international semantic web conference, Proceedings, Part II, Sydney, 21–25 Oct 2013, pp 277–293. https://doi.org/10.1007/978-3-642-41338-4_18
- Feigenbaum L, Williams GT, Clark KG, Torres E (2013) SPARQL 1.1 Protocol. W3C Recommendation. Online at <https://www.w3.org/TR/sparql11-protocol/>
- Fionda V, Gutierrez C, Pirrò G (2014) The swget portal: navigating and acting on the web of linked data. J Web Sem 26:29–35. <https://doi.org/10.1016/j.websem.2014.04.003>
- Fionda V, Pirrò G, Gutierrez C (2015) Nautilod: a formal language for the web of data graph. TWEB 9(1):5:1–5:43. <http://doi.acm.org/10.1145/2697393>
- Harth A, Hose K, Karnstedt M, Polleres A, Sattler K, Umbrich J (2010) Data summaries for on-demand queries over linked data. In: Proceedings of the 19th international conference on World Wide Web, WWW 2010, Raleigh, 26–30 Apr 2010, pp 411–420. <http://doi.acm.org/10.1145/1772690.1772733>
- Hartig O (2011a) How caching improves efficiency and result completeness for querying linked data. In: WWW2011 workshop on linked data on the Web, Hyderabad, 29 Mar 2011. <http://ceur-ws.org/Vol-813/lidow2011-paper05.pdf>
- Hartig O (2011b) Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: The semantic web: research and applications – 8th extended semantic web conference, ESWC 2011, Proceedings, Part I, Heraklion, 29 May–2 June 2011, pp 154–169. https://doi.org/10.1007/978-3-642-21034-1_11
- Hartig O (2012) SPARQL for a web of linked data: semantics and computability. In: The semantic web: research and applications – 9th extended semantic web conference, ESWC 2012, Proceedings, Heraklion, 27–31 May 2012, pp 8–23. https://doi.org/10.1007/978-3-642-30284-8_8
- Hartig O (2013) SQUIN: a traversal based query execution system for the web of linked data. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2013, New York, 22–27 June 2013, pp 1081–1084. <http://doi.acm.org/10.1145/2463676.2465231>
- Hartig O, Buil-Aranda C (2016) Bindings-restricted triple pattern fragments. In: On the move to meaningful internet systems: OTM 2016 conferences – federated international conferences: CoopIS, C&TC, and ODBASE 2016, Proceedings, Rhodes, 24–28 Oct 2016, pp 762–779. https://doi.org/10.1007/978-3-319-48472-3_48
- Hartig O, Özsu MT (2016) Walking without a map: ranking-based traversal for querying linked data. In: The semantic web – ISWC 2016 – 15th international semantic web conference, Proceedings, Part I, Kobe,

- 17–21 Oct 2016, pp 305–324. https://doi.org/10.1007/978-3-319-46523-4_19
- Hartig O, Pérez J (2016) LDQL: a query language for the web of linked data. *J Web Sem* 41:9–29. <https://doi.org/10.1016/j.websem.2016.10.001>
- Hartig O, Bizer C, Freytag JC (2009) Executing SPARQL queries over the web of linked data. In: The semantic web – ISWC 2009, 8th international semantic web conference, ISWC 2009, Proceedings, Chantilly, 25–29 Oct 2009, pp 293–309. https://doi.org/10.1007/978-3-642-04930-9_19
- Ladwig G, Tran T (2010) Linked data query processing strategies. In: The semantic web – ISWC 2010 – 9th international semantic web conference, ISWC 2010, Revised Selected Papers, Part I, Shanghai, 7–11 Nov 2010, pp 453–469. https://doi.org/10.1007/978-3-642-17746-0_29
- Ladwig G, Tran T (2011) Sihjoin: querying remote and local linked data. In: The semantic web: research and applications – 8th extended semantic web conference, ESWC 2011, Proceedings, Part I, Heraklion, 29 May–2 June 2011, pp 139–153. https://doi.org/10.1007/978-3-642-21034-1_10
- Lynden SJ, Kojima I, Matono A, Nakamura A, Yui M (2013) A hybrid approach to linked data query processing with time constraints. In: Proceedings of the WWW2013 workshop on linked data on the web, Rio de Janeiro, 14 May 2013. <http://ceur-ws.org/Vol-996/papers/lidow2013-paper-07.pdf>
- Mika P, Potter T (2012) Metadata statistics for a large web corpus. In: Proceedings of the 5th linked data on the web workshop (LDOW)
- Miranker DP, Depena RK, Jung H, Sequeda JF, Reyna C (2012) Diamond: a SPARQL query engine for linked data based on the rete match. In: Proceedings of the workshop on artificial intelligence meets the web of data (AImWD)
- Paret E, Van Woensel W, Casteleyn S, Signer B, De Troyer O (2011) Efficient querying of distributed RDF sources in mobile settings based on a source index model. In: Proceedings of the 2nd international conference on ambient systems, networks and technologies (ANT 2011), the 8th international conference on mobile web information systems (MobiWIS-2011), Niagara Falls, 19–21 Sept 2011, pp 554–561. <https://doi.org/10.1016/j.procs.2011.07.072>
- Schaffert S, Bauer C, Kurz T, Dorschel F, Glachs D, Fernandez M (2012) The linked media framework: integrating and interlinking enterprise media content and data. In: I-SEMANTICS 2012 – 8th international conference on semantic systems, I-SEMANTICS'12, Graz, 5–7 Sept 2012, pp 25–32. <http://doi.acm.org/10.1145/2362499.2362504>
- Schmachtenberg M, Bizer C, Paulheim H (2014) Adoption of the linked data best practices in different topical domains. In: Proceedings of the 13th international semantic web conference (ISWC)
- Schmedding F (2011) Incremental SPARQL evaluation for query answering on linked data. In: Proceedings of the second international workshop on consuming linked data (COLD 2011), Bonn, 23 Oct 2011. http://ceur-ws.org/Vol-782/Schmedding_COLD2011.pdf
- Tian Y, Umbrich J, Yu Y (2011) Enhancing source selection for live queries over linked data via query log mining. In: The semantic web – joint international semantic technology conference, JIST 2011, Proceedings, Hangzhou, 4–7 Dec 2011, pp 176–191. https://doi.org/10.1007/978-3-642-29923-0_12
- Umbrich J, Hose K, Karnstedt M, Harth A, Polleres A (2011) Comparing data summaries for processing live queries over linked data. *World Wide Web* 14(5–6):495–544. <https://doi.org/10.1007/s11280-010-0107-z>
- Umbrich J, Karnstedt M, Hogan A, Parreira JX (2012) Hybrid SPARQL queries: fresh vs. fast results. In: The semantic web – ISWC 2012 – 11th international semantic web conference, Proceedings, Part I, Boston, 11–15 Nov 2012, pp 608–624. https://doi.org/10.1007/978-3-642-35176-1_38
- Vander Sande M, Verborgh R, Van Herwegen J, Mannens E, Van de Walle R (2015) Opportunistic linked data querying through approximate membership metadata. In: The semantic web – ISWC 2015 – 14th international semantic web conference, Proceedings, Part I, Bethlehem, 11–15 Oct 2015, pp 92–110. https://doi.org/10.1007/978-3-319-25007-6_6
- Van Herwegen J, Verborgh R, Mannens E, Van de Walle R (2015) Query execution optimization for clients of triple pattern fragments. In: The semantic web. Latest advances and new domains – 12th European semantic web conference, ESWC 2015, Proceedings, Portoroz, 31 May–4 June 2015, pp 302–318. https://doi.org/10.1007/978-3-319-18818-8_19
- Verborgh R, Vander Sande M, Hartig O, Van Herwegen J, De Vocht L, De Meester B, Haesendonck G, Colpaert P (2016) Triple pattern fragments: a low-cost knowledge graph interface for the web. *J Web Sem* 37–38: 184–206. <https://doi.org/10.1016/j.websem.2016.03.003>

Linked Data Models

► Graph Data Models

Linked Data Query Processing

► Linked Data Management

Linked Geodata

► [Linked Geospatial Data](#)

Linked Geographic Data

► [Linked Geospatial Data](#)

Linked Geospatial Data

Manolis Koubarakis¹, Konstantina Bereta¹, Charalampos Nikolaou², and George Stamoulis¹

¹Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece

²Department of Computer Science, University of Oxford, Oxford, UK

Synonyms

[Linked geodata](#); [Linked geographic data](#)

Definitions

Linked geospatial data is data about geographic objects made available on the Web using linked data technologies such as RDF and SPARQL and interlinked with other Web data to increase its value for users and applications.

Overview

Huge amounts of geospatial data have been made freely available recently on the Web, for example, maps from geospatial search engines like Google Maps, images from satellites, open geospatial data from national cartographic agencies, and user-contributed geospatial content from social networks. This article surveys the state of the art in the area of linked geospatial data, i.e., geospatial data made available on the Web using linked

data technologies such as RDF and SPARQL and interlinked with other Web data to increase its value for users and applications.

Key Research Findings

The research contributions of academia and industry in the area of linked geospatial data include data models and query languages, implemented systems, visual user interfaces, and applications. These contributions are surveyed in the rest of this article.

Data Models and Query Languages

Early works in the area of linked geospatial data include papers by Perry, Kolas, Koubarakis, and their colleagues (Perry et al. 2006, 2011; Kolas and Self 2007; Koubarakis and Kyzirakos 2010). Based on these early papers, two extensions of SPARQL were defined in 2012: the Open Geospatial Consortium (OGC) standard GeoSPARQL for querying geospatial data expressed in RDF and the stSPARQL language for querying linked geospatial data that changes over time. In 2013, RDFⁱ, a general extension of the RDF data model for representing incomplete information was introduced (Nikolaou and Koubarakis 2013, 2016). RDFⁱ can model incomplete geospatial information using vocabulary introduced by GeoSPARQL, so it is also covered below.

GeoSPARQL

The GeoSPARQL standard defined by OGC (2012) provides vocabulary (classes, properties, and functions) that can be used in RDF graphs and SPARQL queries to represent and query geospatial data. GeoSPARQL follows a modular design and consists of the following components:

- *Core*. This component defines top-level classes that provide users with vocabulary for modeling geospatial information. The classes offered by this component are geo:SpatialObject and geo:Feature. Class geo:SpatialObject is the top

- class defined by GeoSPARQL and has as instances everything that can have a spatial representation. Class `geo:Feature` is the class of all features; *feature* is the term used for geographic objects in the literature of geographic information systems (GIS). Class `geo:Feature` can be a superclass of any class of features users might want to define. Class `geo:SpatialObject` is a superclass of `geo:Feature` and `geo:Geometry` (to be defined in the geometry extension component below).
- *Topology vocabulary extension.* This component provides vocabulary for asserting and querying topological relations between spatial objects. Three families of topological relations are supported coming from the large body of previous research on this topic by GIS, databases, and artificial intelligence researchers. These families are the topological relations for simple features defined in the OGC standard “OpenGIS Simple Feature Access – Part 2: SQL Option” (e.g., `geo:sfTouches`), the Egenhofer topological relations defined by Egenhofer and Franzosa (1991) (e.g., `geo:ehMeet`), and the RCC-8 relations defined by Randell et al. (1992) (e.g., `geo:rcc8ec`). An important point here is that these topological relations can relate not just geometries but also features. They can be asserted by a triple of an RDF graph (e.g., `dbpedia:Athens geo:sfWithin dbpedia:Greece`) but also used in a triple pattern of a SPARQL query (e.g., `?x geo:sfWithin dbpedia:Greece`). The query rewriting extension discussed below provides a mechanism to derive topological relations between features from the topological relations that are satisfied by their corresponding geometries. In this way, GeoSPARQL caters to users that are more interested in qualitative spatial reasoning applications (Randell et al. 1992) but also to more traditional GIS or DBMS users interested in geospatial computations.
 - *Geometry extension.* This component provides vocabulary for asserting and querying

information about geometries. A crucial design decision of GeoSPARQL is to use *literal values* to encode geometries as a single unit and introduce the new data types `sf:wktLiteral` and `gml:gmlLiteral` for these literals. The former data type enables the serialization of geometries using the OGC standard well-known text (WKT) while the latter using the Geography Markup Language (GML).

The geometry extension also defines the class `geo:Geometry` as the superclass of all geometry superclass. It also defines properties for representing metadata of geometries (e.g., `geo:dimension` that captures the topological dimension), for associating features with geometries (e.g., `geo:hasGeometry`) and for associating geometries with their literal serializations (`geo:asWKT` and `geo:asGML`). In addition, this extension defines functions for performing non-topological operations on geometries (e.g., `geof:distance`, `geof:buffer`, `geof:convexHull`, etc.). These functions, which can be used in SPARQL queries, are taken from the OGC standard “OpenGIS Simple Feature Access – Part 2: SQL Option”. • *Geometry topology extension.* This component provides three families of Boolean functions that operate on geometries and can be used in a SPARQL query to check whether given topological relationships hold for these geometries. These Boolean functions correspond to the topological relations of the topology vocabulary extension presented above (simple features, Egenhofer, and RCC-8) and can be applied to geometry literals encoded in WKT or GML (e.g., the Boolean function `geof:ehMeet` that corresponds to the Egenhofer topological relation `geo:ehMeet` mentioned above). In addition, the extension defines the general function `geof:relate` that can be used to specify any topological relation one might be interested in that can be expressed using the Dimensionally Extended 9-Intersection Model (Clementini et al. 1993).

- *RDFS entailment extension.* This component provides a mechanism for realizing the RDFS entailments that follow from the RDFS class and property axioms that hold for the classes and properties introduced by GeoSPARQL and presented in the extensions above.
- *Query rewrite extension.* This component provides a collection of Rule Interchange Format (RIF) rules that derive any of the topological relations of the topology vocabulary extension between two pairs of spatial objects (features or geometries), whenever the corresponding Boolean function of the geometry topology extension holds between the corresponding geometry literals. As an example, the component has a RIF rule named `geor:sfWithin` that can be used to derive the triple `dbpedia:Athens geo:sfWithin dbpedia:Greece` from the fact that the Boolean function `geof:sfWithin` returns true when evaluated with arguments the WKT encodings of the geometry literals corresponding to features `dbpedia:Athens` and `dbpedia:Greece`. The name of the extension (query rewrite) comes from one of its possible implementations which would be to rewrite a given SPARQL query with a triple pattern involving a topological relation between two features, into one that checks whether the corresponding Boolean function holds between the geometry literals corresponding to the features.

stRDF/stSPARQL

The model stRDF is an extension of RDF for representing geospatial information that changes over time (Kyzirakos et al. 2012; Bereta et al. 2013). Similarly to GeoSPARQL, stRDF uses literals of data types `strdf:WKT` and `strdf:GML` to represent geometries serialized using the OGC standards WKT and GML. The data type `strdf:geometry` is also introduced as the union of data types `strdf:WKT` and `strdf:GML`.

The time dimension of stRDF is used to model the *valid time* of triples originally introduced by

the pre-RDF knowledge representation language Telos (Mylopoulos et al. 1990) and rediscovered by Semantic Web researchers (Gutierrez et al. 2007). stRDF allows the use of quads of the form (s, p, o, t) to model the information that t is the time when the fact represented by the RDF triple (s, p, o) is valid in reality. Valid times can be encoded by using literals of the new data type `strdf:period`; these literals represent points or intervals with closed or open endpoints which are themselves literals of type `xsd:dateTime`.

The query language of the model stRDF is stSPARQL (Kyzirakos et al. 2012; Bereta et al. 2013). stSPARQL is an extension of SPARQL 1.1 with essentially the same topological and non-topological functions for simple features introduced by the geometry and the geometry topology extension of GeoSPARQL (only the name space changes: `strdf` instead of `geof`). These functions can be used in the SELECT, FILTER, and HAVING clause of a SPARQL 1.1 query. In addition to these functions, stSPARQL offers four topological functions based on the bounding box of the involved regions (e.g., `strdf:mbbIntersects`), four cardinal direction relations (e.g., `strdf:above`), and three aggregate functions (`strdf:union`, `strdf:intersection`, and `strdf:extent`). Since stRDF supports quads, stSPARQL also allows quad patterns (s, p, o, t) in queries. In addition, stSPARQL offers the following families of temporal functions: functions corresponding to the 13 interval-to-interval relations of Allen (1983) and three point-to-interval relations of Meiri (1996) (e.g., `strdf:after`), period constructors (e.g., `strdf:period_intersect`), and temporal aggregates (e.g., `strdf:maximalPeriod`).

Comparing GeoSPARQL with stSPARQL, we can see that stSPARQL offers slightly more geospatial functionality than the geometry and geometry topology extensions of GeoSPARQL but also temporal functionalities not offered by GeoSPARQL.

RDFⁱ

The RDFⁱ framework (Nikolaou and Koubarakis 2013, 2016) (where “i” stands for “incomplete”)

is an extension of RDF for representing values of properties that exist but are either unknown or partially known. To model this kind of incomplete information, RDF^i extends RDF with the ability to define a new kind of literals for each data type. These literals are called *e-literals* (where “e” comes from the word “existential”) and are allowed to appear only in the object position of triples. RDF^i allows partial information regarding property values represented by e-literals to be expressed by a quantifier-free formula of a first-order constraint language \mathcal{L} . The modeling capabilities of RDF^i have been demonstrated by Nikolaou and Koubarakis (2016) with a number of constraint languages \mathcal{L} that allow RDF^i to express (i) incomplete information corresponding to *marked nulls*, (ii) incomplete temporal information, (iii) spatial properties of abstract or polygonal regions of \mathbb{Q}^2 , and (iv) spatial and metric constraints involving rectangles with sides parallel to the axes in \mathbb{Q}^2 .

RDF^i extends the concept of an RDF graph to the concept of an RDF^i database which is a pair (G, ϕ) , where G is an RDF graph possibly with e-literals and ϕ is a quantifier-free formula of \mathcal{L} . The semantics of RDF^i is inspired by the seminal work of Imieliński and Lipski (1984) and assigns to an RDF^i database (G, ϕ) an infinite set of RDF graphs. Each such graph corresponds to a possible state of the real world represented syntactically by the RDF^i database (G, ϕ) . Query answering for SPARQL queries and RDF^i databases (G, ϕ) amounts to computing the set of certain answers, that is, those answers that are present in the result of the evaluation of the query over each possible RDF graph corresponding to (G, ϕ) . It turns out that the data complexity of checking whether an answer is certain increases from LOGSPACE (the upper bound of evaluating SPARQL graph patterns over RDF graphs) to CONP-complete for all of the aforementioned constraint languages. This result is in line with similar complexity results for querying incomplete information in relational databases.

Comparing GeoSPARQL with RDF^i , we can see that both proposals allow for the assertion of geometric and topological information of features, but only RDF^i is able to reason about the

topology of features when geometrical information about them is missing. The kind of spatial information modeled in RDF^i can be also modeled in appropriate spatial extensions of description logics and the Web Ontology Language (OWL). Closest to RDF^i are the approaches choosing to separate terminological knowledge expressed in some description logic or OWL from spatial knowledge expressed as data assertions (Wessel and Möller 2009). Other approaches follow a tighter integration of terminological and spatial knowledge by equipping description logics with a spatial concrete domain that offers the reasoning capabilities of RCC-8 (Lutz and Milicic 2007). Finally, approaches similar to GeoSPARQL extend the vocabulary of OWL with the topological relations of RCC-8 and propose a model for the representation of qualitative spatial information (Batsakis and Petrakis 2010). In contrast to GeoSPARQL, the semantics of spatial relations is encoded as a set of rules, while reasoning in this extension of OWL can be realized by off-the-self OWL reasoners.

Implemented Systems

One of the very early works in the area of geospatial RDF stores is the implementation described by Brodt et al. (2010) which is built on top of RDF-3X. It does not offer GeoSPARQL support, but it allows for the representation of geometries as WKT typed literals. It also supports spatial operators defined in the OGC standard “OpenGIS Simple Feature Access – Part 2: SQL Option” and it uses an R-tree index.

Another early work is described by Perry et al. (2011) in which the language SPARQL-ST was proposed. SPARQL-ST is a spatial and temporal extension of the SPARQL query language, and it considers spatial and temporal variables and filters, respectively. The query engine for SPARQL-ST was implemented on top of Oracle 10g.

Parliament was the first GeoSPARQL implementation and it supports the following components of GeoSPARQL: core, topology vocabulary, geometry, geometry topology, and RDF entailment. It also supports multiple coordinate reference systems (CRS) and uses an R-tree for indexing geometries.

Strabon is the state-of-the-art spatiotemporal triple store (Kyzirakos et al. 2012; Bereta et al. 2013). It supports both stSPARQL and a subset of GeoSPARQL (core, geometry extension, and geometry topology extension). Strabon is built by extending the well-known triple store Sesame and uses a spatially enabled DBMS as back end (currently PostGIS and MonetDB are supported, but PostGIS is the DBMS used by default).

USeekM is an open-source spatial plugin for Sesame developed by Open Sahara. It supports the following GeoSPARQL components: core, topology vocabulary, geometry, geometry topology, and RDFS entailment. GraphDB is another commercial RDF store that offers GeoSPARQL support by integrating part of the USeekM geospatial functionalities. USeekM uses an R-tree index, while GraphDB uses an Apache Lucene index, which is a general purpose index that also includes spatial support.

There are also systems with geospatial functionalities that offer limited or zero GeoSPARQL support. AllegroGraph is a commercial quad store that offers limited geospatial functionalities: it implements only a few spatial operators, and it supports only point geometries. OpenLink Virtuoso also includes a few geospatial features, for example, it supports point geometries serialized as typed literals and a small subset of the equivalent operations defined in SQL/MM but does not have support for GeoSPARQL. Finally, Stardog is another system with limited GeoSPARQL support in its

commercial edition. It supports point geometries natively. Polygons are supported optionally, if the user configures the system to load the JTS library. It supports only a few spatial operators (`geof:relate`, `geof:distance`, `geof:within`, `geof:nearby` and `geof:area`).

Despite the existence of many geospatial RDF stores, especially after the establishment of GeoSPARQL as an OGC standard, a geospatial ontology-based data access (OBDA) system did not exist until the creation of Ontop-spatial, the geospatial extension of OBDA system Ontop. Ontop performs on-the-fly SPARQL-to-SQL translation on top of relational databases using ontologies and mappings (Calvanese et al. 2017). Mappings express how relational data can be translated into RDF terms, and they can be encoded in the W3C standard R2RML. Ontop-spatial extends Ontop by performing on-the-fly GeoSPARQL-to-SQL translation on top of a spatially enabled DBMS (e.g., PostGIS, SpatiaLite) (Bereta and Koubarakis 2016).

Finally, another system with GeoSPARQL support is Oracle Spatial and Graph. It supports multiple coordinate reference systems, and the recent version Oracle 12c release 2 contains also support for on-the-fly GeoSPARQL-to-SQL translation using R2RML mappings without needing to materialize the data as RDF triples.

Table 1 shows an overview of the geospatial functionality supported by geospatial RDF stores, and Table 2 shows the respective functionalities

Linked Geospatial Data, Table 1 Functionality of geospatial RDF stores and OBDA systems

System	Language	Index	Geometries	CRS support	Relation families in geometry topology
Strabon	stSPARQL, GeoSPARQL	R-tree over GiST	WKT, GML	Yes	Simple features (SF), Egenhofer, RCC-8
Parliament	GeoSPARQL	R-tree	WKT, GML	Yes	SF, Egenhofer, RCC-8
Brodt et al. (RDF-3X)	SPARQL	R-tree	WKT	No	SF
Perry	SPARQL-ST	R-tree	GeoRSS, GML	Yes	RCC-8
Oracle spatial and graph	GeoSPARQL	R-tree	WKT, GML	Yes	SF, Egenhofer, RCC-8
USeekM	GeoSPARQL	R-tree over GiST	WKT	No	SF, Egenhofer, RCC-8
GraphDB	GeoSPARQL	Lucene	WKT	No	SF, Egenhofer, RCC-8
Ontop-spatial	GeoSPARQL	R-tree	WKT, GML	No	SF, Egenhofer, RCC-8

Linked Geospatial Data, Table 2 Functionality of geospatial RDF stores offering limited geospatial support

System	Language	Index	Geometries	CRS support	Available functions
AllegroGraph	Extended SPARQL	Distribution sweeping technique	2D point geometries	Partial	Buffer, bounding box, distance
Virtuoso	Extended SPARQL	R-tree	2D point geometries (in WKT)	Yes	SQL/MM (subset)

offered by systems that provide limited geospatial support.

Geographica is a benchmark that was developed in order to evaluate geospatial RDF stores. The results of the experimental evaluation described by Garbis et al. (2013) show that Strabon is not only a rich geospatial RDF store in terms of functionality, but it is also the most efficient. Strabon, in most cases, outperforms USeekM, Parliament, and Virtuoso, as well as three commercial triple stores with geospatial support (due to license considerations, the names of these commercial systems are not disclosed). In the area of OBDA systems, Ontop-spatial was evaluated by Bereta and Koubarakis (2016), where it was compared with Strabon. The results showed that, in most cases, Ontop-spatial outperforms Strabon because it is able to generate SQL queries that can be very efficiently executed by the underlying geospatial DBMS.

Visual User Interfaces

In addition to the storage and querying systems presented above, many interesting visualization systems for linked geospatial data have been proposed. Some follow the Linked Data Visualization Model principles (Klímek et al. 2014; Thellmann et al. 2015), while others rely on vocabularies (Mutlu et al. 2013; Atemezing and Troncy 2014). All systems support visualization for geospatial data, but only one (Nikolaou et al. 2015) provides support for GeoSPARQL/stSPARQL.

The Linked Data Visualization Model (LDVM) (Brunetti et al. 2013) allows the connection of different datasets with various visualizations in a dynamic way. The process follows a four-stage workflow: source data,

analytical abstraction, visualization abstraction, and view. This model has been adopted by Payola (Klímek et al. 2014) that provides a refined implementation of the LDVM. It consists of various analyzers for automatically classifying datasets and transformers for mapping the data to visualization abstractions. Users can connect to a SPARQL endpoint, load their data, and perform analysis and visualization on a Web browser.

Another tool is CODE Vis Wizard (Mutlu et al. 2013) that relies on the RDF Data Cube Vocabulary, a W3C standard for representing statistical data as RDF. Vis Wizard is a Web-based system that is able to analyze multiple datasets using brushing and linking methods. Similarly, LDVizWiz (Atemezing and Troncy 2014) uses a semiautomatic way to produce possible visualizations of a dataset. LDVizWiz performs ASK queries to categorize data in order to understand the features of a dataset, but this requires adaptation of the queries to new vocabularies when they are detected.

LinkDaViz (Thellmann et al. 2015) is another approach in the same context as the previous two. This framework uses heuristic data analysis and a visualization model in order to automatically bind data and visualization options. The workflow guides the user through the process of visualizing data and starts with the exploration of a dataset. After the user has selected the part of the dataset to be visualized, a ranked list of recommended visualizations is computed. When one of the recommendations is selected, the resulting visualization is displayed, ready for customization.

While most of the visualization tools rely on the ontology of the datasets to produce views, some allow the use of SPARQL queries and vi-

sualize the results. Sextant (Nikolaou et al. 2015) is an open-source visualizer that also supports GeoSPARQL and stSPARQL. The features that distinguish Sextant from other Semantic Web visualization tools are the ability to create, share and edit thematic maps, by combining information from different SPARQL endpoints and enrich them with other well-adopted geospatial file formats, like KML, GML, GeoJSON and GeoTIFF, as well as visualizing the temporal dimension of data. Sextant heavily utilizes Semantic Web technologies, but the user-friendly interface, combined with features like predefined queries, allows both domain experts and non-experts to handle this tool.

Applications

The linked geospatial data technologies discussed above have been used in many interesting applications, e.g., wild fire monitoring (Koubarakis et al. 2013), developing semantic catalogues for Earth observation data archives (Karpasiotaki et al. 2014; Espinoza-Molina et al. 2015), maritime security (Brüggemann et al. 2016), and precision farming (Burgstaller et al. 2017). In the context of these applications, many Earth observation and geospatial datasets have been made available on the Web as linked open data therefore increasing the potential for development of more interesting applications in the future.

Cross-References

- ▶ [Big Data Visualization Tools](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Query Languages](#)
- ▶ [Linked Data Management](#)
- ▶ [Visualization](#)

References

- Allen JF (1983) Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843
- Atemezing GA, Troncy R (2014) Towards a linked-data based visualization wizard. In: Proceedings of the 5th international workshop on consuming linked data
- Batsakis S, Petrakis EGM (2010) SOWL: spatio-temporal representation, reasoning and querying over the semantic web. In: Proceedings of the 6th international conference on semantic systems
- Bereta K, Koubarakis M (2016) Ontop of geospatial databases. In: Proceedings of the 15th international semantic web conference, pp 37–52
- Bereta K, Smeros P, Koubarakis M (2013) Representation and querying of valid time of triples in linked geospatial data. In: Proceedings of the 10th international conference on the semantic web: semantics and big data, pp 259–274
- Brodt A, Nicklas D, Mitschang B (2010) Deep integration of spatial query processing into native RDF triple stores. In: Proceedings of the 18th ACM SIGSPATIAL international symposium on advances in geographic information systems, pp 33–42
- Brüggemann S, Bereta K, Xiao G, Koubarakis M (2016) Ontology-based data access for maritime security. In: Proceedings of the 13th international conference on the semantic web: latest advances and new domains, pp 741–757
- Brunetti JM, Auer S, González RG, Klímek J, Necaský M (2013) Formal linked data visualization model. In: Proceedings of the 15th international conference on information integration and web-based applications & services
- Burgstaller S, Angermair W, Niggemann F, Migdall S, Bach H, Vlahopoulos I, Savva D, Smeros P, Stamoulis G, Bereta K, Koubarakis M (2017) LEOPatra: a mobile application for smart fertilization based on linked data. In: Proceedings of the 8th international conference on information and communication technologies in agriculture, food and environment
- Calvanese D, Cogrel B, Komla-Ebri S, Kontchakov R, Lanti D, Rezk M, Rodriguez-Muro M, Xiao G (2017) Ontop: answering SPARQL queries over relational databases. *Semantic Web* 8(3):471–487
- Clementini E, Di Felice P, van Oosterom P (1993) A small set of formal topological relationships suitable for end-user interaction. In: Proceedings of the 3rd international symposium on advances in spatial databases, pp 277–295
- Egenhofer MJ, Franzosa RD (1991) Point set topological relations. *Int J Geogr Inf Syst* 5(2):161–174
- Espinoza-Molina D, Nikolaou C, Dumitru CO, Bereta K, Koubarakis M, Schwarz G, Datcu M (2015) Very-high-resolution SAR images and linked open data analytics based on ontologies. *IEEE J Sel Topics Appl Earth Observ Remote Sens* 8(4):1696–1708
- Garbis G, Kyzirakos K, Koubarakis M (2013) Geographica: a benchmark for geospatial RDF stores. In: Proceedings of the 12th international semantic web conference, pp 343–359
- Gutierrez C, Hurtado CA, Vaisman AA (2007) Introducing time into RDF. *IEEE Trans Knowl Data Eng* 19(2):207–218
- Imielinski T, Lipski W Jr (1984) Incomplete information in relational databases. *J ACM* 31(4):761–791. ISSN:0004-5411

- Karpathiotaki M, Dogani K, Koubarakis M, Valentin B, Mazzetti P, Santoro M, Di Franco S (2014) Semantic search for Earth observation products using ontology services. In: Proceedings of the 8th international conference on web reasoning and rule systems, pp 173–178
- Klímek J, Helmich J, Necaský M (2014) Application of the linked data visualization model on real world data from the Czech LOD cloud. In: Proceedings of the workshop on linked data on the web
- Kolas D, Self T (2007) Spatially-augmented knowledge-base. In: Proceedings of the 6th international semantic web conference and 2nd asian semantic web conference, pp 792–801
- Koubarakis M, Kyriakos K (2010) Modeling and querying metadata in the semantic sensor web: the model stRDF and the query language stSPARQL. In: Proceedings of the 7th extended semantic web conference on the semantic web: research and applications, pp 425–439
- Koubarakis M, Kontoes C, Manegold S (2013) Real-time wildfire monitoring using scientific database and linked data technologies. In: Proceedings of the joint 2013 EDBT/ICDT conferences, pp 649–660
- Kyriakos K, Karpathiotakis M, Koubarakis M (2012) Strabon: a semantic geospatial DBMS. In: Proceedings of the 11th international semantic web conference, pp 295–311
- Lutz C, Milicic M (2007) A tableau algorithm for description logics with concrete domains and general TBoxes. *J Autom Reason* 38(1–3):227–259
- Meiri I (1996) Combining qualitative and quantitative constraints in temporal reasoning. *Artif Intell* 87 (1–2):343–385
- Mutlu B, Höfler P, Sabol V, Tschinkel G, Granitzer M (2013) Automated visualization support for linked research data. In: Proceedings of the I-SEMANTICS'13 posters & demonstrations track, pp 40–44
- Mylopoulos J, Borgida A, Jarke M, Koubarakis M (1990) Telos: representing knowledge about information systems. *ACM Trans Inf Syst* 8(4):325–362
- Nikolaou C, Koubarakis M (2013) Incomplete information in RDF. In: Proceedings of the 7th international conference on web reasoning and rule systems, pp 138–152
- Nikolaou C, Koubarakis M (2016) Querying incomplete information in RDF with SPARQL. *Artif Intell* 237:138–171
- Nikolaou C, Dogani K, Bereta K, Garbis G, Karpathiotakis M, Kyriakos K, Koubarakis M (2015) Sextant: visualizing time-evolving linked geospatial data. *J Web Semant* 35:35–52
- Open Geospatial Consortium OGC (2012) GeoSPARQL – a geographic query language for RDF data. OGC Implementation Standard, Sept 2012
- Perry M, Hakimpour F, Sheth AP (2006) Analyzing theme, space, and time: an ontology-based approach. In: Proceedings of the 14th ACM international symposium on geographic information systems, pp 147–154
- Perry M, Jain P, Sheth AP (2011) SPARQL-ST: extending SPARQL to support spatiotemporal queries. In: Geospatial semantics and the semantic web – foundations, algorithms, and applications, pp 61–86
- Randell DA, Cui Z, Cohn AG (1992) A spatial logic based on regions and connection. In: Proceedings of the 3rd international conference on principles of knowledge representation and reasoning, pp 165–176
- Thellmann K, Galkin M, Orlandi F, Auer S (2015) Link-DaViz – automatic binding of linked data to visualizations. In: Proceedings of the 14th international semantic web conference, pp 147–162
- Wessel M, Möller R (2009) Flexible software architectures for ontology-based information systems. *J Appl Logic* 7(1):75–99

L

Location Analytic Architecture

► Architectures

Logical Reasoning

► Automated Reasoning

M

Machine Learning

- ▶ [Flood Detection Using Social Media Big Data Streams](#)

Machine Learning Benchmarks

- ▶ [Analytics Benchmarks](#)

Machine Learning for Health and Life Sciences

- ▶ [Big Semantic Data Processing in the Life Sciences Domain](#)

Management of Time

Ted Dunning
MapR Technologies, Santa Clara, CA, USA

Definitions

The management of time in streaming systems refers to how streaming computations make use of the explicit and implicit times associated with messages used in the computation.

Background

In a streaming data system, the data consists of messages that enter the system and then undergo various operations such as filtering, transformation, and aggregation. Between processing steps, these messages are transported by message streams such as those provided by Apache Kafka (Apache Software Foundation 2016c). When messages come from disparate sources via different paths, they may be reordered, delayed, or even lost entirely due to propagation delay or hardware or software failures. Processing steps themselves may also delay, reorder, or lose messages due to multi-threading or various kinds of faults in processing.

It is desirable that the processing of these messages be as reliable and timely as possible, but delay, loss, and reordering of messages can make this difficult or even impossible. In fact, the desire for timeliness is often in conflict with the desire for accuracy forcing a less than perfect trade-off.

Over the life of a message, there are implicit times associated with each step of processing of the message starting with the creation or ingestion of the message and continuing on until the message is no longer of interest and is discarded. These times may not be explicitly recorded, but they still can be considered to exist, at least conceptually. There may also be times associated with the real-world events that may have given rise to a message. Many processing steps make use of different times associated

with messages, particularly for creating what are known as windowed aggregates. Such aggregates are themselves data streams containing messages with a value for each distinct window along with information describing the window.

For instance, we might want to count the number of sales made in each month. To do so, we need to know, of course, about all sales that are made, but we also need to know when each sale is made so that we can credit it to the correct month. Moreover, we can't know the total for any month until it is certain that there are no more sales to be reported. When there is a delay in reporting sales in some month, there must be a corresponding delay before we can report the precise total of sales for that month. In addition, if the reports of sales are delivered out of order with respect to the time the sale occurred, we may have to accumulate counts for multiple months since sales for the new month will be reported even though some sales have not yet been reported for the previous month or months.

The desire for timely results can conflict with the desire for accurate results. Taking the sales example again, it might be that a report could be delayed by as much as a month, even though delays of more than a day or two are quite rare. In this case, we could report a total for a month a few days after the end of the month that is usually correct. A result that is "usually correct" might well be good enough, especially if we can put a bound on how wrong the result might possibly be. It might also be acceptable to report a corrected total the following month as long as the correction is small relative to the originally reported value. Clearly, we need some guide to know when we have seen enough data to output an aggregated value.

Foundations

From the example of sales reports, it is clear that there are many events associated with a sale that could be said to be the moment when the sale occurred. This could be when a handshake is made, when the customer signs the order, when the order is approved by the vendor, when the

order is shipped, or even when full payment is received. It is quite plausible that many or even most sales could involve events that span multiple months. For the purposes of computing some particular quantity, however, it is customary to appeal to some accounting standard to arbitrarily define one of these events as the key point in time that a sale is made. Nothing prevents using a different time in a different computation for a different purpose, but for any given computation, it is typical to define an "event time" when it is said that the event represented by a message in a stream actually happened, but it should be recognized that this definition is entirely specific to a particular user requirement.

There are other times that are much more associated with the computational process itself. For instance, the moment that the message has been received is typically known as "ingestion time," and the moment when the message is actually processed is often known as "processing time." It should be noted that of these three times, only event time is independent of when a computation is done. As such, only computations that are defined in terms of event time are likely to be acceptable to users who need a result that can be reliably reproduced and audited. Such users will care little or not at all about details such as when a computation is run.

In most situations, it is also important that a computation produce precisely the same result if it is run more than once. This is typically only possible if the result is defined in terms of event time since ingestion and processing times are a function of when a program is run. Some stream processing systems such as Apache Storm (Apache Software Foundation 2013) conflated these different concepts, at least initially, and only allowed programs to be expressed in terms of processing time.

It should be noted, however, that computing with event time has its own risk since event time is typically defined outside the streaming system and thus errors in event time are outside the control of the streaming system. Times that are measured automatically are often subject to large systematic errors due to incorrect clock settings. Manually entered times are subject to

large errors due to typographical mistakes. For whatever reason, large errors in times seem one of the most common measurement error. Input validation as early as possible is necessary, and compensation for common errors such as bad clock settings and drained batteries is often necessary. Ingestion time may be subject to similar errors especially if ingestion is highly distributed, but such errors are typically more susceptible to correction. Processing time is least suspect in this regard but also generally least useful except for computing time differences. In general, all externally generated times should be carefully validated as early as possible in a streaming computation. Times in the future, times far in the past, and times on messages where message delay apparently changes discontinuously and by a large amount are all highly suspect.

Generally, computation on a stream is composed of three kinds of steps:

Stateless transformations and filtering of individual records Without any retained state between records, this can be done without any reference to event or any other time at all.

Re-ordering of records, such as sorting by event time Since the input is unbounded, a general sort is impossible. If the disordering of the input is limited, however, then the input can be sorted by event time with bounded storage, computation, and delay.

Computation of a function of a group of records known as a window Since streams are unbounded, windows are typically only useful if the records in a window are more or less localized in time. The grouping of records is known as windowing, and the computed values are called windowed aggregates.

In the context of time management, only sorting and windowed aggregates are of interest because stateless transformations can be done entirely without reference to time.

Watermarks

The use of event time raises some additional complications, however, because messages in a

streaming system are subject to variable and unpredictable amounts of delay, which can disorder records. Computations framed purely in terms of processing time have the great advantage that there is no need to consider the problem of delayed data. Considering only processing time, however, makes it nearly impossible to precisely reproduce a computation. Bringing event time into the conversation necessarily requires consideration of the problem of delay and that, in turn, makes sorting and windowing difficult because it isn't always clear how much messages may have been delayed.

It is often possible, however, to put a bound on how much delay there may have been. Rather than express this in terms of delay, it is handier to keep track of how much of the input stream is complete at any given time. That is, we want to know an event time that is guaranteed to be before the event time in any messages that we may receive in the future from a particular message stream. This time is known as the watermark for the stream. Clearly, if we know that a stream has a certain maximum possible delay, then the current time less this delay is a viable, although, usually, overly conservative, watermark for the stream. Usually we can do better than this and have a safe watermark that is much closer to the current time. Modern systems such as Google's Dataflow (Akidau et al. 2015), Apache Beam (Apache Software Foundation 2016a), and Apache Flink (Friedman and Tzoumas 2016; Apache Software Foundation 2016b) all associate watermarks with input streams. Apache Spark's structured streaming (Apache Software Foundation 2016d) component uses a single maximum delay for an entire computation to compute an estimated watermark for all input streams.

The point of a watermark is that we can know when it is safe to emit any windowed aggregates because the watermark will tell which windows may have pending messages and which windows cannot. Any window that is entirely older than the watermark for a stream will never receive any new data, and thus the corresponding aggregate value will never change. That means that once the watermark passes the end of a window, we can safely emit the current value of any

corresponding aggregates without worrying they will be superseded. If a watermark is precise, it is also clear that this is the best we can do if we want to only emit final values for aggregates and we want to do that as soon as possible.

It is also possible to define a watermark with a weaker guarantee that simply gives a non-zero bound for how much data might still be received with event times before the watermark. Such weak watermarks allow provisional values of aggregates to be emitted that are close to the final value.

Practically speaking, watermarks are often based on heuristic summaries of past system behavior. If messages are only transported using ordered streams and messages are coming from a known set of sources, then each source can inject heartbeat messages into the message stream and depend on the ordering of the stream itself to guarantee that no message is ever reordered behind a heartbeat from the same source. A watermark for the overall input consisting of multiple such streams can be computed by remembering most recently received heartbeat for each source stream. The earliest heartbeat time of any source is then the desired watermark for the composite.

Sorting by Event Time

Sorting messages by event time is the simplest “interesting” thing to be done with disordered streaming data. Messages in a stream subject to delay can be sorted if the delay is bounded, but unbounded delay implies unbounded required storage for temporary values. If watermarks are available and the delay is relatively small, sorting is relatively easy. All messages with event time after the current watermark can be kept in a sorted buffer such as a heap. As the watermark advances in time, messages that now have an event time before the watermark can be emitted in sorted order. The watermark will eventually pass every record because of the bounded delay, so the sort will eventually emit all messages (in order).

Obviously, the practicality of this procedure depends to a great degree on how much delay there might be in the stream since that determines

how many values need to be kept in the sort buffer either in memory or on disk.

Windows and Triggers

Windows specify how messages in a streaming system are grouped for aggregation. For instance, messages might be grouped hourly. Or messages from individual users might be grouped by day in terms of each user’s local time zone. The aggregation on a window can be as simple as a count of the number of records.

Since the input in a streaming system is unbounded, windows are necessary to bound the computation of aggregated values. This is because in a streaming system, any aggregate for a window that is not bounded in time can never be emitted. Different kinds of windows have different characteristics. Some kinds of windows allow different windows to overlap in time. Some kinds of windows reuse messages in multiple windows. Some kinds of windows are defined purely by event time without respect to the other content of messages. In all cases, however, the key point of any windowing definition is that it defines which messages belong to which windows.

Tuple-Based Windows

The simplest form of windowing is to retain the last n values seen. This can be valuable for summarizing gross measurements of overall activity, especially for alerting on system malfunctions. The time since the latest sample is a good way to detect missing data and the n -th last sample is useful in looking for changes in activity rate since it allows an optimum trade-off between detection time and false-positive rate. Tuple-based windows are commonly used for systems monitoring.

Tuple-based windows are defined independently of event time and are thus not usually suitable for producing aggregates for external consumers.

Time-Based Windows

Another common windowing scheme is to group messages by either processing time or event time. Grouping on processing time can be useful for monitoring the state of the streaming system itself, while grouping on event time is more

useful when analyzing the data instead of the streaming system. Though not strictly required, it is common that each window is characterized by a start time and an end time and all messages with event times greater than or equal to the start time and less than the end time will belong to that window. Commonly, such windows are of mostly constant duration and have start times separated by a constant stride. If the stride is equal to or greater than the duration, then there is no overlap in the windows. With a shorter stride, windows will overlap and messages will belong to multiple windows.

Complications can arise in this simple picture, especially when calendars are involved. For instance, a window that starts at the beginning of each month and ends at the end obviously has a variable stride and duration since months vary in length. Windows that start at the beginning of each day can have the same kind of problem due to daylight savings time. Leap seconds can cause similar, though much smaller, surprises.

Complex Time-Based Windows

Basing window membership of messages on the event time of each message independently is insufficient for some kinds of computation. The most prominent example of this are session-based windows in which successive messages are assigned to the same window as long as the time between messages is less than a specific threshold. To further complicate matters, session windows are also typically user specific, so an enormous number of windows can be active at the same time.

Complex windows such as user-specific sessions can be surprisingly resilient to certain data errors such as large offsets in event time, especially if such errors are consistent on a source by source basis.

Data-Defined Windows

In general, messages in a stream can be grouped into windows by stateful rules defined using any aspect of the messages being grouped. All of the windowing schemes discussed so far here can be defined as special cases of this. Support for general data-defined windows is still quite un-

common for streaming analytics systems. Apache Beam and Apache Flink are examples of systems that support data-defined windows.

Triggers

In the context of stream processing, windows define how messages are to be grouped together for aggregation but do not specify when the computation of an aggregate must happen. Triggers, on the other hand, define when such aggregations should actually be computed but are independent of the window definition itself. Ideally, the aggregated results for each window would be computed precisely when the last message in a window is processed. In practice, such a prompt computation may not be convenient or efficient. Delaying a bit so that many aggregates can be computed together at the same time might, for instance, make better use of computing resources.

Another good illustration of the value of alternative triggering of a window is session windows. For many uses of session windows, it is sufficient to only check every 10 min or so to see if any sessions have expired. This can be done using a single timer to periodically invoke a scan of all active windows, preferably in order of last contact. Detecting expired sessions exactly when they expire, on the other hand, typically requires a timer per active window and causes frequent small computations which are less efficient than larger but rarer computations.

Differences Between Time in Streams and Time in Databases

Temporal databases emphasize the processing of time values that represent and distinguish concepts such as when a fact is valid (valid time) and when a fact is known (transaction time). For instance, on the first of April, we may learn that Mark Twain was born in 1965 and died in 2003. Seven months later, in November, we might conclude after extensive investigation that he was actually born in 1835 and died in 1910.

Our first fact from April fool's day stated that Mark Twain's life spanned 1965–2003 but that "fact" had a validity period from April until November. The more authoritative (and actually correct) fact that he lived from 1835 to 1910 had a

validity period from November onward. Expressing the evolution of knowledge in a database can be very important in financial and legal settings where you may have to answer questions of the form “what did you know and when did you know it.”

In a streaming system, the situation is made a bit more complex by the fact that many big data systems are global in extent, and thus there is no single time when data is valid since there is typically no single database that is the repository of all truth. On the other hand, streaming does make things simpler as well since it is difficult for a database to store all old states, whereas a streaming system inherently does exactly that as long as all messages are retained. That means the state of what was known at any time in history can be recreated precisely if one can replay messages in historical order up to the desired point in time.

In spite of the difference, there is still a useful analogy to be drawn between transaction time in a temporal databases and event time in streaming systems. Likewise, valid time in a temporal database and processing time in a streaming system have an analogical function. The streaming concepts in each case are more general since a single message may have multiple event times associated with it as well as multiple processing times.

Cross-References

- ▶ [Adaptive Windowing](#)
- ▶ [Apache Apex](#)
- ▶ [Apache Flink](#)
- ▶ [Apache Kafka](#)
- ▶ [Apache Samza](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Stream Processing Languages and Abstractions](#)
- ▶ [Types of Stream Processing Algorithms](#)

References

- Akida T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E, Whittle S (2015) The dataflow model: a practical approach to balancing correctness,

latency, and cost in massive-scale, unbounded, out-of-order data processing. In: Proceedings of the VLDB endowment, vol 8, pp 1792–1803

Apache Software Foundation (2013) Storm. <http://storm.apache.org>

Apache Software Foundation (2016a) Beam. <http://beam.apache.org>

Apache Software Foundation (2016b) Flink. <http://flink.apache.org>

Apache Software Foundation (2016c) Kafka. <http://kafka.apache.org>

Apache Software Foundation (2016d) Spark structured streaming. <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>

Friedman E, Tzoumas K (2016) Introduction to Apache flink: stream processing for real time and beyond. O'Reilly Media, Inc., Sebastopol

Mapping

- ▶ [Schema Mapping](#)

MapReduce

- ▶ [Hadoop](#)

Maximum Eccentricity in Real-World Networks

- ▶ [Degrees of Separation and Diameter in Large Graphs](#)

Measure

- ▶ [Metrics for Big Data Benchmarks](#)

Measured System

- ▶ [System Under Test](#)

Media Lifespan

- ▶ [Data Longevity and Compatibility](#)

Metadata

- ▶ [Flood Detection Using Social Media Big Data Streams](#)

Metadata Extraction

- ▶ [Data Profiling](#)

Metadata Generation

- ▶ [Data Profiling](#)

Metrics for Big Data Benchmarks

Alain Crolotte
Teradata Corporation, El Segundo, CA, USA

Synonyms

[Figure of merit](#); [Gauge](#); [Measure](#); [Yardstick](#)

Definitions

A big data (BD) benchmark metric is a standard measure indicating the adequacy and cost-effectiveness of the system under test (SUT) to perform a particular big data task or set of tasks.

Overview

The following section provides a survey of the big data benchmark and associated metrics evolution over the years until present day. In the foundations section we list all the main metrics while the following sections present the definitions and properties for each metrics category namely, response time and throughput, availability and reliability, price-performance and system-level metrics. We then go over the various methods to aggregate these individual metrics while the criticisms section reviews the metrics surveyed. We then list the key applications of these benchmarks. Finally cross-references and references are provided.

Historical Background

The need for computer performance metrics was identified as early as 1985 (Anon 1985) and led to debit-credit, the first benchmark with a standard and associated metrics, namely, throughput and price performance. The debit-credit benchmark was then used and extended by the Transaction Processing Performance Council (TPC) [www.tpc.org provides detailed information on all the TPC benchmarks] and became TPC-A followed by TPC-B and TPC-C, which had similar metrics: tpmC (transactions per minute in benchmark C) as the performance metric and dollars(tpmC) as the price-performance metric. Following the development of OLTP benchmark standards, the TPC started decision support benchmarks in the early 1990s, first TPC-D then TPC-H and finally TPC-DS. While there are differences between these benchmarks, the associated performance metrics are similar, namely, throughput in terms of queries per hour (QPH) and price performance. While the OLTP benchmarks involved only one test, the TPC decision support benchmarks involve several tests each associated with its own metric requiring an aggregation rule to define a final single performance metric.

Wikipedia defines big data as “a term for data sets that are so large or complex that traditional data processing application software is inade-

quate to deal with them” – big data therefore poses a challenge addressed as early as 2005 by the Apache Hadoop framework with its HDFS distributed file system and its tool suite including map reduce (MR). Initially contained in the kit were micro-benchmarks. These were WordCount, grep, Sort, Terasort (which became TPCx-HS), and in later releases, micro-benchmarks addressing the performance of its components such as the HDFS file system or the underlying network architecture or MR. For these built-in Hadoop benchmarks a result set are produced showing well-defined metrics for each test or subtest, e.g., bytes written or read per second. Unlike in the case of standard benchmarks such as the TPC benchmarks, users can aggregate and use the individual micro-benchmark metrics as they please. In the same vein, a number of Hadoop benchmark suites have been developed. An example is HiBench (Huang et al. 2010) which is a superset of the Hadoop-provided micro-benchmarks with targeted add-on workloads such as machine learning. Each domain or sub-domain has its own metric in these cases, but unlike for industry-standard benchmarks, there is no attempt to define a single performance metric.

Big data was eventually characterized by the 3Vs (volume/velocity/variety) (Laney and Deja 2012). The 3Vs became 4Vs and even 5Vs but without adding much to the data “bigness” argument, so we will stay with the 3Vs. TPC decision support benchmarks address volume by recognizing that size matters requiring that the scale factor be part of the metric and disallowing comparisons across scale factors. Those scale factors are bounded by 100 TB which is very small in the big data world as petabytes of data are commonly mentioned. They also address velocity by recognizing the importance of refreshing data, but the amount of new data, the frequency, and often near real-time requirement for BD maintenance functions far exceed what is covered by the TPC decision support benchmarks. Variety, a qualitative metric, is the ability to handle many types of data in addition to structured data accessible by SQL. Such data include semi-structured data such as weblogs or unstructured data such as text. The first integrated end-to-end big data

analytics benchmark including the 3Vs was BigBench (Ghazal et al. 2013), an extension of TPC-DS with semi-structured data (weblogs), unstructured data (product reviews), and 30 analytical queries representing the realities of decision support in a modern enterprise. Initially BigBench was implemented on Teradata Aster with SQL and SQL-MR. BigBench was proposed by the TPC as TPCx-BB with a complete kit typically running on Hadoop. The performance metrics for this benchmark are also a form of throughput combining the load time, the timing for the power test, and the timing of the throughput test and price/performance. The allowed scale factors in TPCx-BB follow the same pattern as those in TPC-DS, but the upper bound is now 1 petabyte. Like A major rework of BigBench was proposed that brings important variety improvements and incorporates the concept of late binding (Ghazal et al. 2017). The metrics for BigBench V2 are the same as those for TPCx-BB.

In Ivanov et al. (2015), a detailed survey of 20 BD benchmarks along with the associated metrics is provided. Time is by far the most popular metric in the form of query time or execution time closely followed by throughput and price performance. New metrics appeared such as client latency, data processed per second, and data processed per joule. Newer BD benchmarks addressing new data types (for instance, in BigBench V2 the semi-structured data is in JSON format) but also new structures such as streams or graphs may require the creation of new metrics. For instance, Graph 500 (<http://graph500.org/>), a public-domain benchmark addressing graph-based systems, introduces TEPS defined as the number of edge traversals per second yet another form of throughput. The field of BD benchmarks is an active area of research, but metrics are mainly aimed at measuring performance, and the only two dimensions of performance are time and accuracy.

Foundations

There are many BD benchmarks available: industry standard, public domain, or company-

sponsored. The associated BD benchmarks metrics, however, fall into broad categories. In Han et al. (2017), the authors propose a classification of BD benchmark metrics in which they distinguish between user-perceivable and non-observable performance metrics. Observable performance metrics include (1) response time, (2) throughput, (3) reliability, and (4) availability. Metrics that are not directly observable by a user are hardware architecture metrics dealing with CPU, memory, and connectivity. BD benchmarks can involve several types of metrics as portrayed in Han et al. (2017) and Ivanov et al. (2015).

Response Time and Throughput

Typical time-based performance measures in the context of computer benchmarks are either elapsed time for a fixed number of operations that are well-defined in advance or number of operations performed in a fixed amount of time. In both cases, by dividing the number of operations by the elapsed time, a throughput measure is obtained. While a typical performance measure is transactions per second, elapsed times are also commonly used. While a lower elapsed time is an indication of higher performance, a higher throughput indicates a higher performance. The performance is always proportional to the amount processed and inversely proportional to the elapsed time.

Availability and Reliability

Accuracy-based performance measures are typically reliability, i.e., the number of successful operations compared to the number of unsuccessful operations, and availability, which measures the percentage of time the system is up and running. All TPC benchmarks involve tests that insure system availability. Several built-in Hadoop micro-benchmarks are designed to stress the system and determine reliability and availability.

Price Performance Metrics

Performance usually comes at a cost or price, so it is important to determine not only the performance but also the cost of the system. Money is an obvious cost metric, but it is not the only

one. Another important price metric is power consumption. This led the TPC to introducing an optional energy component to all its benchmarks. The price performance metric is price divided by performance, e.g., dollars per operations per unit of time, such as dollar per transaction per query per hour. An example of a price performance metric is $$/HSph@SF$, used in TPCx-HS (Terasort). The scale factor SF is part of the metric to avoid comparisons at different scale factors.

System-Level Metrics

For special-purpose benchmarks and benchmarks addressing a large number of workloads such as HiBench and BigDataBench, it is appropriate to look at performance at a system level. Metrics at this level can be CPU utilization breakdown, processor intensity, processor memory level parallelism, or scale-out performance. These metrics do not provide an application-level metric such as a query per hour, but they do provide black box measures of performance that can be used to optimize resource allocation in a data center, for example, a subject addressed by DCBench (<http://prof.ict.ac.cn/DCBench/>).

M

Aggregation of Individual Metrics

In a BD benchmark, there can be several levels of aggregation. The first level involves individual homogeneous transactions, e.g., queries. In a benchmark test, each individual transaction has a measured elapsed time. The transactions can be run sequentially or in parallel – the difference between the end of the last executed transaction and the start time of the first executed transaction is the total elapsed time. To illustrate our point, we will use the TPC-H benchmark that can be considered as a very early BD benchmark. In TPC-H there are two tests: (1) a power test and (2) a throughput test. The power test runs 2 refresh functions and 22 queries sequentially in a prespecified order. The average transaction time during the power test would therefore be the sum of the elapsed times divided by 24 (22 queries + 2 refresh functions). Instead of the usual simple average, the geometric mean was used. The idea of using the geometric mean to

summarize *normalized* benchmark results was proposed in (Fleming and Wallace 1986). Unfortunately it is easy to forget that if the individual values are *raw* (e.g., elapsed times expressed in seconds in the case of TPC-H), then the arithmetic mean is much more appropriate (Crolotte 2009). An argument used by the proponents of the geometric mean is that the arithmetic mean is biased toward large values as a very large individual value in the calculation of the arithmetic mean will “overwhelm” the metric bringing the throughput down. However, the geometric mean has a similar property since an extremely low value will bring the metric down potentially very close to zero which will bring the throughput unrealistically high. These issues are discussed in detail in Crolotte (2009). Aggregating individual test metrics into a single figure of merit will face similar challenges.

Criticisms

The final metric for the TPC-H power test is queries per hour at scale factor (QpH@SF), i.e., SF*3600/g, where g is the geometric mean of the 24 power test elapsed times expressed in seconds and SF the scale factor expressed in gigabytes. Because of the use of the geometric mean, the TPC-H power metric is not a real query per hour, but it has the dimensions of a query per hour. The second test is the throughput test where queries are run in parallel and the metric is based on the total elapsed time which makes it a real query per hour rate. In order to aggregate the two tests, the geometric mean is used again. The use of the geometric mean in TPC benchmarks has been criticized mainly because the formula itself was complicated since the formula for the geometric mean of n values involves the product of those values to the power 1/n. Yet the final metric is scalable and has the dimension of a throughput albeit not a real one, but, as noted in Huppler (2009), this metric is relevant as it does provide a measure of performance for a system running the application on a database of a defined size. In spite of the criticisms about its metric, TPC-D was a very successful benchmark followed by TPC-H which remains a de facto standard

until today. TPC-DS has four performance tests each with an elapsed time, and these four tests are aggregated again with the geometric mean. It remains to be seen whether, for most cases, applying the geometric mean would result in an ordering different from that obtained by applying the arithmetic mean.

Key Applications

Metrics are a powerful way to provide a basis for comparison between systems and implementations on particular systems. However, unless there is a guarantee, the comparison may not be valid. The need for metrics appeared in the mid-1980s and led to the formation of the TPC which provided such a guarantee through a formal process involving well-defined standards, audits by certified auditors, and advertising rules preventing unfounded claims. The TPC benchmarks also come with benchmark kits making it easy to implement workloads quickly. For instance, even before TPC-DS was a full standard, several vendors used the workload and took some of the 99 TPC-DS queries in isolation in order to compare DBMS and Hadoop implementations. This led to many unfounded claims as to one implementation being better than the other and vice versa. When a workload consists of several types of operations, the only fair time measure that can be applied is either the total elapsed time or the average elapsed time per operation.

Response time and throughput are especially suited for comprehensive, end-to-end BD benchmarks. These benchmarks are useful at showing the power of a system to run a particular application, for instance, BigBench. They can also be used to tune a system such as a Hadoop cluster. GridMix (<https://hadoop.apache.org/docs/r1.2.1/gridmix.html>), for instance, is used for performance engineering of a Hadoop cluster by helping to identify bottlenecks and tune job and user priorities. Benchmark suites such as HiBench and its associated metrics are more targeted at evaluating hardware or optimizing Hadoop components especially map reduce.

Cross-References

- ▶ [Analytics Benchmarks](#)
- ▶ [Auditing](#)
- ▶ [Cloud Big Data Benchmarks](#)
- ▶ [Component Benchmark](#)
- ▶ [CRUD Benchmarks](#)
- ▶ [Distributed File Systems](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [Graph Data Management Systems](#)
- ▶ [Hadoop](#)
- ▶ [Microbenchmark](#)
- ▶ [System Under Test](#)
- ▶ [TPC-DS](#)
- ▶ [TPC-H](#)
- ▶ [TPCx-HS](#)

References

- Anon et al (1985) A measure of transaction processing power. Datamation, 31 pp. 112–118, 1 April 1985
- Crolotte A (2009) Issues in benchmark metric selection. In: TPCTC, Lyon, pp 146–152
- Fleming P, Wallace J (1986) How to not lie with statistics: the correct way to summarize benchmark results. Commun ACM 29:218–221
- Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen HA (2013) BigBench: towards an industry standard benchmark for big data analytics. In: SIGMOD
- Ghazal, A., Ivanov, T., Kostamaa, P., Crolotte, A., Voong, R., Al-Kateb, M., Ghazal, W., Zicari, R. (2017) BigBench V2 – the new and improved BigBench. In: SIGMOD
- Han R, Kurian L, Zhan J (2017) Benchmarking big data systems: a review. IEEE Trans Serv Comput, (99): 1–18
- Huang S, Huang J, Dai J, Xie T and Huang B (2010) The HiBench benchmark suite: characterization of the mapreduce-based data analysis. In: ICDEW, Mar 2010
- Huppler K (2009) The art of building a good benchmark. In: Performance evaluation and benchmarking, vol 5895. Springer, Berlin/Heidelberg, pp 18–30
- Ivanov T, Rabl T, Poess M, Queralt A, Poelman J, Poggi N, Buell J (2015) Big data benchmark compendium. In: 7th TPC technology conference. (TPCTC)
- Laney D (2012) Deja VVVu: others claiming Gartner's construct for big data. Gartner Blog Netw. 14 Jan 2012. <https://blogs.gartner.com/doug-laney/deja-vvvue-others-claiming-gartners-volume-velocity-variety-construct-for-big-data/>

Microbenchmark

Nicolas Poggi

Databricks Inc., Amsterdam, NL, BarcelonaTech (UPC), Barcelona, Spain

Synonyms

[Component benchmark](#); [Functional benchmark](#); [Test](#)

Definitions

A microbenchmark is either a program or routine to measure and test the performance of a single component or task. Microbenchmarks are used to measure simple and well-defined quantities such as elapsed time, rate of operations, bandwidth, or latency. Typically, microbenchmarks were associated with the testing of individual software subroutines or lower-level hardware components such as the CPU and for a short period of time. However, in the BigData scope, the term microbenchmarking is broadened to include the cluster – group of networked computers – acting as a single system, as well as the testing of frameworks, algorithms, logical and distributed components, for a longer period and larger data sizes.

M

Overview

Microbenchmarks constitute the first line of performance testing. Through them, we can ensure the proper and timely functioning of the different individual components that make up our system. The term *micro*, of course, depends on the problem size. In BigData we broaden the concept to cover the testing of large-scale distributed systems and computing frameworks. This chapter presents the historical background, evolution, central ideas, and current key applications of the field concerning BigData.

Historical Background

Origins and CPU-Oriented Benchmarks

Microbenchmarks are closer to both hardware and software testing than to competitive benchmarking, opposed to application-level – macro – benchmarking. For this reason, we can trace microbenchmarking influence to the hardware testing discipline as can be found in Sumne (1974). Furthermore, we can also find influence in the origins of software testing methodology during the 1970s, including works such as Chow (1978). One of the first examples of a microbenchmark clearly distinguishable from software testing is the *Whetstone benchmark* developed during the late 1960s and published later by Curnow and Wichmann (1976).

The *Whetstone benchmark* is considered the first general-purpose benchmark suite that helped later to set industry standards in computer performance (Longbottom 2017). The first implementations of the benchmark were composed of a set of *ALGOL* and *FORTRAN* routines with the intention of comparing computer architectures and optimizing compilers (Longbottom 2017), in this case, by measuring the number floating point instructions per second of a system. *LINPACK* is another representative microbenchmark example from the late 1970s (Dongarra et al. 2003). *LINPACK* is still in use primarily in its parallel version – *HPL* – in high-performance computing (HPC) environments (A. Petitet 2004) and in the supercomputing Top500 list (TOP500.org 1993).

Another example of a benchmark with a long track of revisions and still in use today is the family of SPEC CPU benchmarks. *SPEC CPU* was standardized and is maintained by the Standard Performance Evaluation Corporation (SPEC 1989), and it is frequently used in official and academic publications (Nair and John 2008; Cantin and Hill 2001). Another early benchmark is UnixBench (1983), in this case, a suite or collection of different benchmarks including those from the Whetstone to graphics card microbenchmarks.

Disk and Network

Unlike HPC and CPU benchmarking, we can characterize BigData – at least the open source-based frameworks, as being GNU/Linux-based, 64-bit x86 architecture deployments over commodity hardware. As in BigData, the main bottlenecks were initially found in the storage and networking layers (Poggi et al. 2014). Due to being readily available, and already part of the system administration routines, the first level of benchmarking in a BigData system typically involves using popular Unix testing tools. In this category, we find the network testing tools such as ping (1983) and traceroute (1987) and disk management utilities including dd (1985) and hdparm (1994) available for most common Linux distributions. After network testing tools, we can find system benchmarks developed by the community, most notably in the disk/file system category: *Bonnie* (bonnie circa 1989), one of the first disk benchmarks with a later 64-bit for *Bonnie++* fork (bonnie++ 1999); *IOzone* another file system benchmark utility (iozone 2002); and Flexible I/O Tester (FIO), which was used in 2012 to set the world record in IOPS (benchmark 2012 IOPS world record) and continues to be popular among storage manufacturers. On networking, *netperf* by Hewlett-Packard (HP) (netperf 1995) has been a popular tool during the late 1990s and early 2000s. Another widely used tool involves the different versions of iperf (2003) still widely used in the field.

BigData

In the beginning there was Sorting (Rabl and Baru 2014)

While preceding the BigData era, we can track the influence of the *sort* benchmark in the original MapReduce paper proposed by Google (Dean and Ghemawat 2004). The sort benchmark led by Jim Gray in an industry-academia collaboration was first introduced in 1985 to measure the input-output (I/O) of a computer system (Anon et al. 1985) as part of the *DebitCredit database* benchmark (Rabl et al. 2014). The sort benchmark

was simple: generate and then sort one million, hundred-byte records stored on disk. Since its appearance, different versions – mainly increasing the input size – and the contests (Sortbenchmark.org 1987) organized around the benchmark have greatly influenced the data processing field. The introduction of the sort benchmark in the original MapReduce paper and its simplicity but also its usefulness of stressing I/O components (Poggi et al. 2014) of a system, sort – in its 1-terabyte format – became the defacto standard in big data benchmarks. TeraSort was later standardized by the Transaction Processing Performance Council (TPC) as TPCx-HS in 2014 (Nambiar et al. 2015) as a way to provide a technically rigorous and comparable version of the algorithm. Since the first releases of Hadoop – the open source implementation of MapReduce in Examples (2016) – the Hadoop examples package has been included with the distribution. Hadoop examples originally included the *Grep* (also listed in Dean and Ghemawat (2004)) and *WordCount* sample applications as microbenchmarks. The *WordCount* application has been widely used in educational contexts, becoming one of the first programs a user of Hadoop would implement. *WordCount* is CPU bound (Poggi et al. 2014), in that sense, not as useful as *TeraSort* to stress and benchmark a cluster. Since then, the Hadoop examples has grown to include an increasing number of microbenchmarks and test applications (Noll 2011), most notably the *dfsio* benchmark to measure the Hadoop Distributed File System (HDFS) throughput.

Other notably early Hadoop benchmark included in Hadoop v1 is *GridMix* by Douglas et al. (2007). *GridMix* simulates multiple users sharing a cluster and includes workloads of cache loads, compression, decompression, and job configuration regarding resource usage. The Hadoop microbenchmark suite designed by Islam et al. (2014) helps with detailed profiling and performance characterization of various HDFS operations. Likewise, the microbenchmark suite designed by Lu et al. (2014) provides detailed profiling and performance characterization of

Hadoop RPC over different high-performance networks. Similarly, *HiBench* (Huang et al. 2010) has extended the *dfsio* program to compute the aggregated throughput by disabling the speculative execution of the MapReduce framework. *HiBench* also adds machine learning and web crawling benchmarks to the suite. Similarly, *BigDataBench* from the ICT, Chinese Academy of Sciences (2014) provides a collection of over 33 microbenchmarks in different categories.

As BigData evolves closer to data warehousing and business intelligence (BI), so do the benchmarks. Most likely, the *Wisconsin benchmark* was the first database-like microbenchmark, designed to measure the performance of individual SQL operators (DeWitt and Levine 2008). *MRBench* by Kim et al. (2008) is an early example which provided microbenchmarks in the form of MapReduce jobs derived from *TPC-H* (Transaction Processing Performance Council 2014). The *CALDA* benchmark proposed by Pavlo et al. (2009) was developed to compare Hadoop against parallel database systems. It included five SQL-based queries which could be implemented to the target system. Both the AMPLab and Hivebench benchmarks are based on *CALDA*'s queries. The Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al. 2010) is a set of benchmarks for performance evaluations of key/value pair and cloud data-serving systems. YCSB was originally designed to compare Cassandra, HBase, and PNUTS in a cloud setting to a traditional database (MySQL). Over time, YCSB has been extended to support a variety of data systems (YCSB-Web 2017) and become the de facto benchmark for key/value stores. Also, YCSB is still frequently used in technical conferences such as in Poggi and Grier (2017).

M

Foundations

Most benchmarks can be classified into two categories: micro- and macro-benchmarks. (Waller 2015)

Performance testing is a subset of performance engineering (Jain 1991b; Testing 2017), where microbenchmarking – or simply benchmarking – is one of the main tools employed to verify reliability and speed of a system. Microbenchmarks are the first line of testing tools for determining the correct functioning and configuration of a node, the interconnect (networking), and then the cluster – a group of networked computers acting as one system. A microbenchmark is then either a program or routine to measure and test the performance of a single component, task, or algorithm.

Microbenchmarks are used to measure simple and well-defined quantities such as elapsed time, the rate of operations – throughput, bandwidth, or latency. Typical use cases for microbenchmarks involve comparing algorithms (i.e., *sorting*) or the performance of the same software routine tested in different hardware platforms but using the exact same input as in Waller (2015). Furthermore, microbenchmarks often follow a *white box* approach as they are implemented having the specification of the target system or the application's code available.

While there are several classifications of benchmarks, according to Ehliar and Liu (2004), there are two practical approaches to benchmarking: micro and application-level (macro). Furthermore, Ehliar et al. distinguishes between micro and application-level according to the ease of implementation and the testing of one aspect at a time, as in microbenchmarks, to creating a realistic and useful application (macro). However, other authors such as Seltzer et al. (1999) argue that many micro- and macro-benchmarks do not efficiently model real workloads. Thus part of the dangers of relying only on benchmark results as the only basis for high-level decisions.

Microbenchmarks in BigData

Traditionally microbenchmarks were associated with the testing only of a system or lower-level hardware components such as CPU, main memory, and I/O on a single node. In the networking context, including a couple of nodes or networking devices, but in both cases for a short period

of time. However, in the BigData scope, the term microbenchmarking is broadened to include the cluster of servers as the system under test and to include the testing of the frameworks or algorithms on which data applications are later built on.

Moreover, microbenchmarks in BigData include the testing of virtualized or logical components such as distributed memory and file systems provided by middleware and layers of abstractions at the software level opposed to the lower-level traditional system benchmarks. Within this context, the metrics of interest are either the total processing power, throughput, or bandwidth of the cluster as a whole, usually for a longer span of total test time which can last for several days (Poggi et al. 2014) and large input data sets to fully stress the total capacity of the system.

Properties of a Good Microbenchmark

On the topic of benchmarking, (Gray 1992) has identified four import criteria for a successful benchmark: relevance, portability, scalable, and simple. In particular, for microbenchmarks, simplicity is the main priority as it must perform a single, well-defined task, with reproducible results. In Gregg (2013), the author makes an emphasis of simplicity being the main advantage of microbenchmarks. In BigData and cloud computing, microbenchmarks also need to be scalable in both weak and strong scaling. Strong scaling refers to keeping the problem size – the data set – fixed but increasing the number of processing elements, typically the number of nodes or containers. In weak scaling, the data set is increased in proportion to the number of processing units. The weak scaling property of the microbenchmark is of particular importance to BigData, as it provides a way to increase the volume of data to be processed or scale factor as in TPC (2017) benchmarks.

In practice, when benchmarking weak scaling properties, the volume of data is also increased without adding more processing capacity, to measure in part the stability of the system and the rate in which performance decreases. Some examples of weak vs. strong scaling can be found in the

ALOJA project results (ALOJA-Web 2015). The property of portability in the BigData field is usually covered as most widely used frameworks, i.e., Hadoop and Spark, are based on the Java virtual machine (JVM), which is multi-platform compatible. Furthermore, GNU/Linux, Hadoop, and Spark are open sourced, as well as most of the aforementioned benchmarks. Relevance is the most challenging feature of a microbenchmark. Testing only a single task or component, a microbenchmark often does not represent a real-world application or use case. A microbenchmark can also misrepresent a system or measure an incorrect property.

The Dangers of Microbenchmarks

Several authors have raised the importance and of good practices in benchmarking and about the dangers of using only microbenchmarks for decision-making. In Traeger et al. (2008), authors perform a comprehensive evaluation of over 106 storage benchmark research papers. Traeger et al. (2008) claim that most benchmarks available at the time were flawed and that statistical analysis of results was mostly incomplete. Gregg (2013) adds to the dangers of benchmarking coining the term “benchmarking sins.” The list of *sins* includes casual benchmarking, benchmark faith, testing the wrong thing, using complex tools, cheating, and many statistical and numerical errors while benchmarking. In Jain (1991a), the author expands the list of “common mistakes in benchmarking,” including not taking into account the overheads of monitoring tools and not measuring transient performance. Traeger et al. (2008) recommend using a combination of several micro- and at least a macro-level benchmark should be run on the target system to understand better the performance.

Microbenchmarks, however, can excel in the repeatability, portability, speed of execution, and automation of the benchmarks as tests. For these reasons, microbenchmarks are often part of Continuous Integration (CI) development tools to detect performance regressions in new pieces of code.

Microbenchmarks in the Development Cycle

Microbenchmarks are often written as part of software development and testing cycle. When developing new functionality, the programmer may decide – most often not – to write a test that measures the performance of the new code. This process usually occurs alongside the coding of unit or functional code tests. As to performance tests – microbenchmarks – usually take longer to execute than unit tests; the tests might not be run with every iteration of the CI suite. For this reason, the performance tests are likely to be run manually and often is the case in practice that is not run frequently after the new feature is finished and shipped.

A relevant example of microbenchmarks in the development and testing process can be found in the Apache Spark’s SQL project source code repository (Apache Spark 2015). The benchmark’s folder contains a *base class* to encapsulate adding and running new benchmarks a set of microbenchmarks. The test classes embed the data generation step when needed and output the CPU architecture and time details, as well as the rate of operations achieved. These tests are meant to be run in the developer workstation or CI environment, as they are often single node or even single-threaded – opposed to a full Spark cluster setting. Moreover, most of these tests are marked as *ignored* in the code, which prevents the CI and testing such as *maven* or *sbt* tools to run them by default.

Key Applications

It was mentioned in the Historical Background Section that data processing benchmarks started with the *DebitCredit* database benchmark and specifically the included *sort* test (Rabl and Baru 2014). We can trace the influence of *sort* to the original MapReduce publication (Dean and Ghemawat 2004) and its *TeraSort* variant included in the Hadoop examples and the TPCx-HS standard (Nambiar et al. 2015). *TeraSort* is still frequently used to benchmark new cluster installations.

This section focuses on a selection of microbenchmarks that are still highly relevant to the industry and are part of the BigData Benchmarking Compendium by Ivanov et al. (2015) and the works of Rabl and Baru (2014) and Poggi et al. (2014). Here benchmarks are classified into system, functional, and database categories. Furthermore, several other early key microbenchmarks where mentioned such as Hadoop *Grid-Mix* and *MRBench*. Other benchmarks that have their own section in this publication, such as HiBench, SparkBench, and YCSB, are briefly discussed.

System

Disk and Filesystem Tools

For data processing, in particular, the storage – disk and file systems – is one of the most important resources to carefully design and tune to achieve high performance. Here the focus is on simple GNU/Linux tools that can be used to test that the target system can achieve a stable and desired operation. For input/output (I/O), the metrics of interest are bandwidth measured in megabytes-per-second MB/s, latency in microseconds, and the number of I/O operations per second or IOPS. IOPS and latency are metrics that are gaining importance (Poggi and Grier 2017) as BigData evolves from batch to more interactive use cases, i.e., streaming, also, to hardware advances such as flash-based storage technologies, e.g., SSDs and NVMe. The task of disk and network testing is usually assigned to system administrators while the functional and database to application roles. This situation leads to different teams not sharing the results or repeating the tests in the future, a situation which can lead to decreased performance over time.

dd (dd 1985) is a disk utility found in most Unix-like operating systems. Its primary purpose is to create, convert, and copy files. However, it can also be used for benchmarking raw disk devices and file systems, as the tool has multiple options and flags both for reading and writing files. Most notably, it contains options to access the raw devices directly and to bypass OS caches. In both cases, it performs only sequential operations

– opposed to random R/W. The sample output below is an example of a mounted networked volume from a popular cloud provider (Store 2017):

```
# dd sample output for writing
      10GB of data from EBS
10000+0 records in
10000+0 records out
10485760000 bytes (10 GB) copied ,
   83.3s , 126MB/s

#dd sample output for reading
      10GB of data from EBS
10000+0 records in
10000+0 records out
10485760000 bytes (10 GB) copied ,
   82.8s , 126MB/s
```

It is interesting to note that in this case, the drive wrote and read at the same rate of 126 MB/s. In this case, the volume is a networked-attached SSD. Magnetic drives (HDD) usually present a lower write than reading rate.

Flexible I/O Tester (FIO) (benchmark 2012 IOPS world record): FIO is the most widely used tool by drive manufacturers to measure device specifications. It was originally written by Jens Axboe to automate the testing of Linux schedulers and I/O subsystem. Since then it has been ported to several operating systems outside GNU/Linux including BSD, Solaris, and Windows. FIO allows testing file systems or raw devices directly. Also to do sequential and random read and write tests. Multiple-file reading and writing are also possible, as well as selecting the number of jobs, I/O queue depth, I/O library, and request (payload) size.

In addition to *dd* and *FIO*, which are single-node tools, cluster-oriented tools are needed to measure cluster-aggregated bandwidth for reading and writing files. For example, for Hadoop's HDFS, the *dfsio* microbenchmark is often used.

Networking

For networking, the most widely used tool is *iperf* (iperf 2003). *Iperf* can be started in client or server mode to measure the bandwidth and latency between two nodes. It has the limitation of only accepting connections from one client. To benchmark all of the nodes at the same time – and

stressing the switching infrastructure – several servers and clients need to be started pointing to different ports. Using this technique, it allows to test the whole cluster, but the aggregated metrics need to be calculated separately from the individual *iperf* reports.

Functional

Hadoop Examples and HiBench

On the BigData functional level, the most widely used benchmarks include the examples of the Hadoop examples package included in most Hadoop distributions. The examples have evolved from including only *grep* and *WordCount* to over 15 sample jobs to test a specific feature of the cluster or to aid in Hadoop’s framework development. Running Hadoop examples is discussed in depth in Noll (2011) blog. Also, benchmarking platforms such as ALOJA by Poggi et al. (2014) enables to easily run a selection of tests in a target cluster, as well as providing results of benchmarks ran in several architectures. In Hadoop versions 2.7 and higher, the example applications have been split into *MapReduce* and *common* packages. The continuously growing list of supported tests can be found at the examples repository (2018). Below is a list of the most representative benchmarks included and some of the extensions found in the first version of *HiBench* (Huang et al. 2010). *HiBench* started as a convenient package and extension to the Hadoop examples but has evolved into a multi-framework and custom benchmarking suite.

- **TeraSort** sorts 1TB of data generated by the TeraGen program distributed with Hadoop. TeraSort is widely used as a reference in research papers as well as in BigData competitions. TeraSort is I/O and CPU intensive.
- **Wordcount** counts the number of word occurrences in large text files. It is distributed with Hadoop and used in many MapReduce learning books. It is CPU bound.
- **Sort** uses the MapReduce framework to sort the input directory into the output

directory, being predominately I/O and shuffle intensive.

- **DFSIO** (or Enhanced DFSIO in HiBench) is an I/O-intensive benchmark to measure throughput in HDFS using MapReduce. It features separate read and write benchmarks.
- **Pagerank**, an implementation of Google’s web PageRanking algorithm. It crawls Wikipedia sample pages.

Additional benchmarks included in HiBench:

- **Bayes** Bayesian Machine Learning classification using the Mahout library. The input of this benchmark is extracted from a subset of the Wikipedia dump.
- **K-means** Mahout’s implementation of the k-means algorithm for knowledge discovery and data mining.
- **HiveBench** the OLAP-style Join and Aggregation queries are adapted from the CALDA benchmark (Pavlo et al. 2009). HiveBench has the goal to test the Hive SQL performance.

M

SparkBench

SparkBench (Li et al. 2015), developed by IBM, is a comprehensive Spark-specific benchmarking suite. The purpose of SparkBench is to analyze tradeoffs between different configurations in Spark and to guide system tuning. It is composed of over ten workloads in four categories including machine learning, graph processing, streaming, and SQL queries. SparkBench reports two metrics: *job execution time* (seconds) and *data process rate* (MB/second).

Database

Besides hardware and software testing, BigData benchmarking is heavily influenced by data processing and databases in general. Moreover, currently it is sometimes difficult to differentiate between a BigData system such as Spark with its SQL library and a massively parallel processing (MPP) database. Over time, what previously were considered functional (Chen et al. 2012) or even application-level benchmarks such as database benchmarks such as the *DebitCredit* (Anon et al. 1985) and the *Wisconsin* benchmark (DeWitt

and Levine 2008; Chen et al. 2012); by extracting specific queries, they can also be considered microbenchmarks. An early example of this fusion is the *CALDA* benchmark, a simple benchmark used initially to compare Hadoop to a distributed database. Over time, CALDA (and its variants) became the de facto benchmark for the early SQL-on-Hadoop (Poggi et al. 2016) systems such as Hive and Spark, with the later AMPLab (AMPLab 2013) adaptations of CALDA.

CALDA – Pavlo’s Benchmark and Adaptations

Pavlo’s benchmark (Pavlo et al. 2009; Stonebraker et al. 2010) consists of five tasks defined as SQL query testing scans, joins, aggregations, and a user-defined function (UDF). The benchmark was developed to specifically compare the capabilities of Hadoop with those of commercial parallel relational database management systems (RDBMS) but was used as a base to also compare newer emerging frameworks such as Hive and Spark.

AMP Lab BigData Benchmark extended CALDA to measure the analytical capabilities of BigData warehousing solutions in general. The hosted benchmark results currently provide quantitative and qualitative comparisons of five data warehouse systems: RedShift, Hive, Stinger/Tez, Shark, and Impala. It supports different data sizes to scale also in cluster size.

YCSB

The Yahoo! Cloud Serving Benchmark (YCSB) (Cooper et al. 2010; Patil et al. 2011) is a set of benchmarks for performance evaluations of key/value pair and cloud data-serving systems. YCSB was originally designed to compare Cassandra, HBase, and PNUTS in a cloud setting to a traditional database (MySQL). Over time, YCSB has been extended to support a variety of data systems (YCSB-Web 2017) also including MongoDB and Riak and has become the de facto benchmark for key/value stores. Also, YCSB is still frequently used in technical conferences such as in Poggi and Grier (2017).

Concluding Remarks

As final words, there is a fine distinction between micro- and macro-level benchmarks. What initially was presented as a complex, application-like benchmark, over time, as its behavior is understood and technology advances, microbenchmarks are usually extracted from the application, as is the case with BigData microbenchmarks, especially in the database-like properties of the systems. Microbenchmarks today are still the first line of performance testing during development and setting up a production system. Furthermore, the benchmarking field is constantly evolving, and microbenchmarking inheriting functionality from macro-benchmarks and expanding with technology.

Cross-References

- ▶ [Component Benchmark](#)
- ▶ [SparkBench](#)
- ▶ [TPCx-HS](#)
- ▶ [YCSB](#)

References

- ALOJA-Web (2015) Aloja home page. <https://aloja.bsc.es/>
- AMPLab (2013) <https://amplab.cs.berkeley.edu/benchmark/>
- Anon EA, Bitton D, Brown M, Catell R, Ceri S, Chou T, DeWitt D, Gawlick D, Garcia-Molina H, Good B, Gray J, Homan P, Jolls B, Lukes T, Lazowska E, Nauman J, Pong M, Spector A, Trieber K, Sammer H, Serlin O, Stonebraker M, Reuter A, Weinberger P (1985) A measure of transaction processing power. Datamation 31(7):112–118. <http://dl.acm.org/citation.cfm?id=13900.18159>
- Apache Spark (2015) Apache spark repository – benchmarks. <https://github.com/apache/spark/tree/master/sql/core/src/test/scala/org/apache/spark/sql/execution/benchmark>
- benchmark (2012 IOPS world record) F (2005) https://en.wikipedia.org/wiki/Jens_Axboe
- bonnie++ (1999) <https://en.wikipedia.org/wiki/Bonnie++>
- bonnie (circa 1989) <https://www.tbray.org/ongoing/When/200x/2004/11/16/Bonnie64>

- Cantin JF, Hill MD (2001) Cache performance for selected spec cpu2000 benchmarks. ACM SIGARCH Comput Archit News 29:13–18
- Chen Y, Raab F, Katz R (2012) From TPC-C to big data benchmarks: a functional workload model. In: WBDB
- Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Trans Softw Eng SE-4(3):178–187. <https://doi.org/10.1109/TSE.1978.231496>
- Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, SoCC 2010, Indianapolis, 10–11 June 2010, pp 143–154
- Curnow HJ, Wichmann BA (1976) A synthetic benchmark. OLW Comput J 19(1):43–49
- dd (1985) [https://en.wikipedia.org/wiki/Dd_\(Unix\)](https://en.wikipedia.org/wiki/Dd_(Unix))
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design and implementation (OSDI'04). USENIX Association
- DeWitt DJ, Levine C (2008) Not just correct, but correct and fast: a look at one of Jim Gray's contributions to database system performance. SIGMOD Rec 37(2): 45–49. <https://doi.org/10.1145/1379387.1379403>
- Dongarra JJ, Luszczek P, Petitet A (2003) The linpack benchmark: past, present, and future. Concurr Comput Pract Exp 15:2003
- Douglas C et al (2007) Grid-mix. <https://hadoop.apache.org/docs/r1.2.1/gridmix.html>
- Ehliar A, Liu D (2004) Benchmarking network processors. In: Swedish system-on-chip conference
- Examples H (2016) <https://apache.googlesource.com/hadoop-common/+/release-0.1.0/src/examples/org/apache/hadoop/examples>
- examples repository H (2018) <https://github.com/apache/hadoop/tree/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples>
- Gray J (1992) Benchmark handbook: for database and transaction processing systems. Morgan Kaufmann Publishers Inc., San Francisco
- Gregg B (2013) Systems performance: enterprise and the cloud. Always learning, Prentice Hall. <https://books.google.de/books?id=xQdvAQAAQBAJ>
- hdparm (1994) <https://sourceforge.net/projects/hdparm/>
- Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 22nd international conference on data engineering workshops, pp 41–51. <https://doi.org/10.1109/icdew.2010.5452747>
- ICT, Chinese Academy of Sciences (2014) BigDataBench 3.1. <http://prof.ict.ac.cn/BigDataBench/>
- iozone (2002) <https://en.wikipedia.org/wiki/IOzone>
- iperf (2003) <https://en.wikipedia.org/wiki/Iperf>
- Islam NS, Lu X, Wasi-ur Rahman M, Jose J, Panda DKD (2014) A Micro-benchmark suite for evaluating HDFS operations on modern clusters. Springer, Berlin/Heidelberg, pp 129–147. https://doi.org/10.1007/978-3-642-53974-9_12
- Ivanov T, Rabl T, Poess M, Queralt A, Poelman J, Poggi N, Buell J (2015) Big data benchmark compendium. In: TPCTC, pp 135–155
- Jain R (1991a) The art of computer systems performance analysis – techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley-Interscience, New York
- Jain R (1991b) The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley-Interscience, New York
- Kim K, Jeon K, Han H, Kim SG, Jung H, Yeom HY (2008) Mrbench: a benchmark for mapreduce framework. In: 14th international conference on parallel and distributed systems, ICPADS 2008, Melbourne, 8–10 Dec 2008, pp 11–18
- Li M, Tan J, Wang Y, Zhang L, Salapura V (2015) Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM international conference on computing frontiers, CF'15. ACM, New York, pp 53:1–53:8
- Longbottom R (2017) Whetstone benchmark history and results. <https://www.roylongbottom.org.uk/whetstone.htm>
- Lu X, Wasi-ur Rahman M, Islam NS, Panda DKD (2014) A micro-benchmark suite for evaluating Hadoop RPC on high-performance networks. Springer International Publishing, Cham, pp 32–42. https://doi.org/10.1007/978-3-319-10596-3_3
- Nair AA, John LK (2008) Simulation points for SPEC CPU 2006. In: IEEE international conference on computer design, pp 397–403. ISSN:1063-6404
- Nambiar R, Poess M, Dey A, Cao P, Magdon-Ismail T, Qi Ren D, Bond A (2015) Introducing TPCx-HS: the first industry standard for benchmarking big data systems. Springer International Publishing, Cham, pp 1–12. https://doi.org/10.1007/978-3-319-15350-6_1
- netperf (1995) <https://www.cup.hp.com/netperf/NetperfPage.html>
- Noll M (2011) Benchmarking and stress-testing a hadoop cluster. <http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfs-nnbench-mrbench/>
- Patil S, Polte M, Ren K, Tantisiriroj W, Xiao L, López J, Gibson G, Fuchs A, Rinaldi B (2011) YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: ACM symposium on cloud computing in conjunction with SOSP 2011, SOCC'11, Cascais, 26–28 Oct 2011, p 9
- Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M (2009) A comparison of approaches to large-scale data analysis. In: SIGMOD, pp 165–178
- Petitet JDAC A, Whaley RC (2004) Hpl – a portable implementation of the high-performance linpack benchmark for distributed-memory computers. ICL – UTK Computer Science Department Retrieved 22 Sept 2016
- ping (1983) [https://en.wikipedia.org/wiki/Ping_\(networking_utility\)](https://en.wikipedia.org/wiki/Ping_(networking_utility))

- Poggi N, Grier D (2017) Evaluating NVMe drives for accelerating HBase. Dataworks Summit (Hadoop Summit) San Jose
- Poggi N, Carrera D, Vujic N, Blakeley J et al (2014) ALOJA: a systematic study of hadoop deployment variables to enable automated characterization of cost-effectiveness. In: IEEE international conference on big data, big data 2014, Washington, DC, 27–30 Oct
- Poggi N, Berral JL, Fenech T, Carrera D, Blakeley J, Minhas UF, Vujic N (2016) The state of sql-on-hadoop in the cloud. In: 2016 IEEE international conference on big data (big data), pp 1432–1443. <https://doi.org/10.1109/BigData.2016.7840751>
- Rabl T, Poess M, Baru CK, Jacobsen H-A (2014) Specifying big data benchmarks – First workshop, WBDB 2012, San Jose, 8–9 May 2012, and Second workshop, WBDB 2012, Pune, India, 17–18 Dec 2012, Revised selected papers. Lecture notes in computer science, vol 8163. Springer (2014). <https://doi.org/10.1007/978-3-642-53974-9>
- Rabl T, Baru C (2014) Big data benchmarking tutorial. In: IEEE international conference on big data, big data 2014, Washington, DC, 27–30 Oct
- Seltzer M, Krinsky D, Smith K, Zhang X (1999) The case for application-specific benchmarking. In: Proceedings of the seventh workshop on hot topics in operating systems, HOTOS'99. IEEE Computer Society, Washington, DC, pp 102
- Sortbenchmarkorg (1987) <http://sortbenchmark.org/>
- SPEC (1989) <https://www.spec.org/>
- Stonebraker M, Abadi DJ, DeWitt DJ, Madden S, Paulson E, Pavlo A, Rasin A (2010) MapReduce and parallel DBMSs: friends or foes? Commun ACM 53(1): 64–71
- Store AEB (2017) <https://aws.amazon.com/ebs/>
- Sumne FH (1974) Measurement techniques in computer hardware design. State of the Art Report No 18, pp 367–390
- Testing SP (2017) https://en.wikipedia.org/wiki/Software_performance_testing
- TOP500org (1993) https://www.top500.org/project/top500_description/
- TPC (2017) [https://www\(tpc.org](https://www(tpc.org)
- traceroute (1987) <https://en.wikipedia.org/wiki/Traceroute>
- Traeger A, Zadok E, Joukov N, Wright CP (2008) A nine year study of file system and storage benchmarking. Trans Storage 4(2):5:1–5:56. <https://doi.org/10.1145/1367829.1367831>
- Transaction Processing Performance Council (2014) TPC benchmark H – standard specification. Version 2.17.1
- UnixBench (1983) <https://github.com/kdlucas/byte-unixbench>
- Waller J (2015) Performance benchmarking of application monitoring frameworks. BoD – books on demand
- YCSB-Web (2017) Ycsb code repository. <https://github.com/brianfrankcooper/YCSB>

Mobile Big Data: Foundations, State of the Art, and Future Directions

Chii Chang¹, Amnir Hadachi², Satish Narayana Srirama¹, and Mart Min³

¹Mobile & Cloud Lab, Institute of Computer Science, University of Tartu, Tartu, Estonia

²Institute of Computer Science, University of Tartu, Tartu, Estonia

³Thomas Johann Seebeck Institute of Electronics, Tallinn University of Technology, Tallinn, Estonia

Definitions

Mobile Big Data represents the discipline of acquiring, processing and managing various large volume, high velocity data streams derived from heterogeneous connected mobile devices and applications in a reliable and valuable manner.

Overview

Emerging ubiquitous mobile services and applications have unveiled the Mobile Big Data era in which the large volumes of data derived from mobile Internet traffics become valuable sources to support various personal and public services such as personal recommendation services, spatiotemporal event detection, social behavior analytics, network resource planning, and many more. While data scientists are aware of the value of Mobile Big Data, they also need to understand the challenges involved. In general, Mobile Big Data derives from different sources, including the mobile application services, the network services, and the mobile devices themselves. The heterogeneous data sources and the data acquisition paths can influence the quality of data and can further influence the overall performance of the big data services. In order to address the major research trends, this chapter provides a state-of-the-art discussion of concept, management technology, and challenges in Mobile Big Data.

Introduction

The increasing Internet-based mobile services and applications have extensively influenced the sources of data traffic in the global Internet. Today, a large amount of Internet data transmission comes from mobile devices for accessing Web contents, file transmissions with cloud storages, participating in online societies, and so forth. Ericsson Research (Ericsson 2017) forecasts that global mobile broadband subscriptions will reach 8.3 billion in the year 2022 and 6.8 billion of these subscriptions derive from smartphones. Further, Cisco forecasts by the year 2021, global mobile data traffic will reach 49 exabytes, and the annual mobile data traffic will exceed half a zettabyte (Cisco 2017). Additionally at the same year, 50% of connected devices in global level will be smartphones (including phablets) (Cisco 2017). Explicitly, this phenomenon has indicated the potential of mobile data and motivated the discipline of *Mobile Big Data*.

Mobile Big Data has three characteristics – spatiotemporal distribution, data aggregation properties, and social correlations (Zhang et al. 2016a) – which grant Mobile Big Data unique features to support extensive services for human society, including personal services and public services. In general, information systems utilize Mobile Big Data for various enhancements such as user-centric recommendations, predicting activities in order to provide proactive services, providing context-aware services based on environmental factors (Cheng et al. 2017), or improving the performance and quality of mobile e-healthcare (Wang et al. 2016). Further, Mobile Big Data also help public service in domains of networking (Zhang et al. 2016a), transportation (Algizawy et al. 2017), and urban living (Zhang et al. 2016b). For instance, by analyzing Mobile Big Data, one is capable to identify the crowd density and the common behaviors of residents in a specific period of time and location.

Similar to the general big data research domain, Mobile Big Data involves six fundamental challenges – *volume*, *velocity*, *variety*, *variability*, *veracity*, and *value*. To enumerate, *volume*

represents the most common descriptor of Big Data, due to the fact that the data is growing in a continuous manner with each day and beyond our capabilities of handling it (Jin et al. 2015); *velocity* involves receiving data streams and the challenge of managing them in real time; *variety* reflects the diversity of the nature of data and its forms; *variability* is related to the inconsistencies and uncertainties of incoming data stream; *veracity* property involves the correctness and authenticity of the data collected; and the *value* reflects the huge value behind Mobile Big Data and its promising market potential.

In order to address the challenges involved in the “6 Vs” mentioned above, system developers need to understand the fundamental concept of Mobile Big Data, where the data comes from, and how their systems manage the data in terms of data acquisition and processing. Accordingly, this chapter covers the foundations of Mobile Big Data, state of the art in management, and open challenges.

This chapter is organized as follows. Section “[The Origins of Mobile Big Data](#)” provides the fundamental concept of Mobile Big Data. Section “[Management of Mobile Big Data](#)” discusses the management technologies of Mobile Big Data. Afterward, we discuss the open challenges of Mobile Big Data in section “[Road Map: Challenges and Future Prospectives](#)”. This chapter is concluded in section “[Conclusion](#)”.

M

The Origins of Mobile Big Data

The staggering growth in the use of mobile devices and their capabilities is driving the production and consumption of Mobile Big Data in many different ways. With all the advancements in sensor technology, mobile devices are loaded with many embedded sensors, such as GPS, accelerometer, gyroscope, and magnetic field. Furthermore, mobile devices have the ability to manage external sensors and interact with or collect data from them. As a result, people and devices are generating large amounts of data related to their interactions with different devices

via the use of telecommunication services and mobile applications or by getting connected to Wi-Fi spots (Cheng et al. 2017). In general, the sources of Mobile Big Data can be resumed into four main categories: (a) application services, (b) mobile devices, (c) telecommunication services, and (d) Wi-Fi access points.

Data from Mobile Application Services

A considerable amount of work has been done, and projects are carried out with the aim of gathering data from mobile application services for research purposes. We can distinguish between two sources of data: internal sensors of a mobile device and data related to the use of mobile applications. For example, the project Mobile Millennium, carried out by the University of California, Berkeley (2008–2010), demonstrated the use of GPS-enabled mobile devices in estimating road traffic status and transit trip planning (Herring et al. 2010). This type of data can also be utilized to understand the end user behavior and its personality as shown in Chittaranjan et al. (2013), where the authors built up a personality profile based on the data collected from the internal sensors and interactions of the end user with the mobile application. The application and the use of this type of data can go beyond our imagination. For example, there are many innovative applications based on this type of data for developing smart transportation (Astarita et al. 2016), urban planning, mapping geographic information, etc.

Data from Mobile Devices

Mobile devices can also collect data from external sensors, due to the technological advancement of wireless devices as illustrated in El Khaddar et al. (2012), where the authors

present emerging wireless technologies for an e-health application. Most of the applications in e-health are focusing on the use of external sensors in collecting or processing data using mobile devices. For example, in Bellairs et al. (2016) the authors demonstrated the capability of transforming a mobile device into a mobile microplate reader by using an external sensor and a mobile application. It is clear that external sensors linked to mobile devices can generate interesting and large amounts of data which is applied in many interdisciplinary research activities, such as personalized medicine (Kwon et al. 2016), Internet of Things (Rathore et al. 2016), etc.

Data from Telecommunication Services

Considerable amount of data is also generated through the interactions between mobile devices and mobile networks. The most frequently used data in research and industrial applications are the call detail records (CDR) and the visitor location registry (VLR). CDR data is a set of information about telecom transactions that operator uses to generate customer billing. The collected records contain the following information (Table 1).

VLR data is a real-time stream data of all the technical logs and events generated by several base station controllers (BSC) in the mobile network. These technical logs are very dense data and contain information such as handover, cell subscribe, cell unsubscribe, packet data, etc. Mobile data has opened new doors for analyzing human mobility from all its aspects and angles. From this perspective, many investigations are done using mobile data in estimating tourists movement and constructing their daily trajectory with a focus on meaningful locations (Sikder et al. 2016). Another example is the use of

Mobile Big Data: Foundations, State of the Art, and Future Directions, Table 1

Example of CDR's data attributes

Attributes	Description
IMSI	International mobile subscriber identity
IMEI	International mobile equipment identity
CellID	The id of the transmitter providing the coverage
Timestamp	The registered time when the event occurred
Event	This is the type of event that triggered the registration of the record such as call, sms, browsing, etc

CDR data for forecasting mobile users movement within the mobile network (Hadachi et al. 2014) and extracting their exact location (Lind et al. 2017a). The idea behind all this analysis is to commence with building a continuous model for describing mobile users mobility patterns, social interactions (Lind et al. 2017b), and daily activities for the purpose of real-time target marketing and for enhancing the mobile users experience within the mobile operator network.

Data from Wi-Fi Access Points

Wi-Fi access points can also be sources of Mobile Big Data. This type of data contains information about devices interactions with Wi-Fi spots, such as fingerprints, signal strength, MAC addresses, etc. The applications developed behind this data are mainly focused on indoor localization and tracking. Commonly, Wi-Fi data can be used for localization in indoor environments (Cheng et al. 2005). In addition, this data can also be used for refining the accuracy of localization in the mobile network (Zandbergen 2009) and in orientation and navigation for emergency evacuation from buildings (Rueppel and Stuebbe 2008).

Management of Mobile Big Data

Mobile Big Data management involves two basic phases – data acquisition and data processing. In general, the data acquisition phase describes where the mobile device delivers its data and where the system stores the data. Further, the data processing phase illustrates where the system executes the data processing activities.

Data Acquisition

Global Centralized

Common mobile information systems collect data from mobile nodes to their global central data center and then provide the processed data to their requesters after the systems complete the data processing. Essentially, collecting data from mobile nodes involves a number of technologies, such as request/response-based approach using HTTP; Internet Engineering Task Force (IETF)

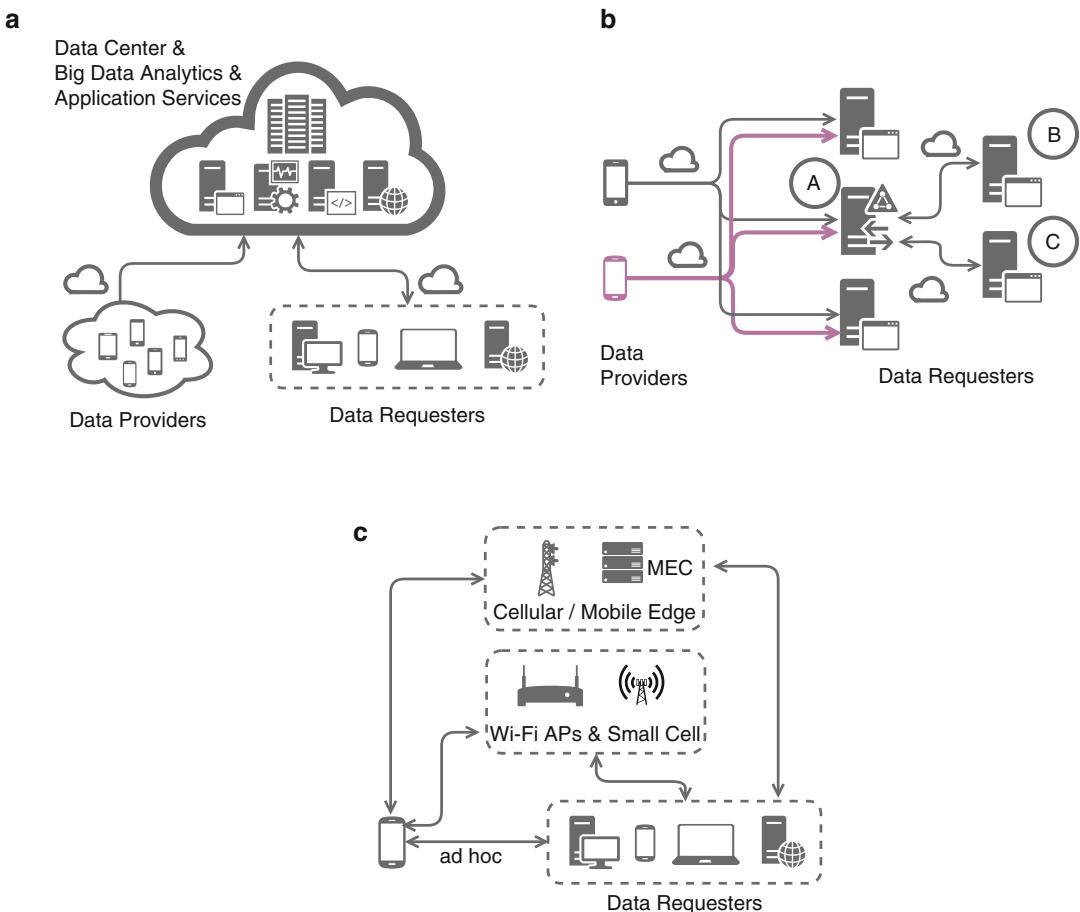
RFC 7252, Constrained Application Protocol (CoAP), TCP, or UDP; and publish-subscribe-based approach using Extensible Messaging and Presence Protocol (XMPP), Advanced Message Queuing Protocol (AMQP; ISO/IEC 19464), or Message Queue Telemetry Transport (MQTT; ISO/IEC PRF 20922). Alternatively, numerous industrial solutions are available, such as open-sourced global-based AllSeen Alliance, AllJoyn framework, and Open Connectivity Foundation (OCF), IoTivity. Centralized data collection is an ideal model for providing global big data services since the data is collected at a single point; with corresponding APIs, third-party service providers can easily request for the data they need. For example in Fig. 1a, data requesters such as application end users or third-party Web service providers may subscribe to the mobile data via the global big data analytics servers.

Global Peer to Peer

In many real-world scenarios, peer-to-peer (P2P) networking provides better efficiency and is more reliable than the centralized model. For example, considering the sensitiveness and the privacy issues, mobile healthcare data is commonly not available to a global repository which is accessible to third parties, and the corresponding mobile device commonly delivers its data directly to a specific receiver (e.g., a hospital data server) in a P2P manner. For another example, recent Internet of Things (IoT) systems also utilize P2P networking based on Blockchain, which is the core technology of the popular Bitcoin system (Nakamoto 2008). Explicitly, P2P-based data acquisition is becoming one of the major models in the Mobile Big Data field. Aside from Blockchain, general technologies to enable P2P-based data acquisition include JXTA technology, Universal Plug and Play (UPnP), BitTorrent protocol, and so forth.

Edge Networks

The edge network-based data acquisition approaches aim to improve the efficiency of data delivery in terms of reducing the data transmission latency and the burden of the global central servers. For example, mobile ad hoc and



Mobile Big Data: Foundations, State of the Art, and Future Directions, Fig. 1 Mobile Big Data acquisition approaches. (a) Data is managed by global data center; (b) mobile nodes have installed multiple service clients to deliver data to different nodes in P2P manner; the

P2P network also utilize data broker nodes to forward data, e.g., node A brokers data to node B and C; (c) mobile devices deliver data to nodes in three types of edge networks

mesh network in social or wireless sensing applications focus on routing the messages among the devices in physical proximity when they are encountered. Further, the mobile and wireless Internet gateway nodes such as Wi-Fi access points and their associated networks, small cell nodes, and cellular base stations are capable of providing Internet-less data delivering. Below, we discuss the three common edge network-based data acquisition models.

Proximity-Based Mobile Peer to Peer

Mobile devices such as smartphones are commonly integrated with Wi-Fi and Bluetooth tech-

nologies. These technologies allow devices to communicate with one another when they encounter in a physical proximity. Essentially, a small group of devices forming a proximity-based mobile P2P (PMP2P) network does not generate much data. However, a recent technology is known as Mobile Mesh Network (MMNET) (Shen et al. 2014) utilizing hybrid connectivity of mobile devices to extend the topology of mobile ad hoc network. It indicates that a peer within the MMNET may receive a large amount of data while it is moving.

Besides the ongoing industrial standard – IEEE 802.11s, which is a Wireless LAN

standard for mesh networking – different parties have introduced their MMNET frameworks. These frameworks include Serval Project (servalproject.org); Open Garden, FireChat (opengarden.com); HypeLabs (hypelabs.io); etc.

Wireless Local Area Network

Today, many areas of the world contain Wi-Fi access point (AP) networks with network virtualization technology, such as Cisco IOS platform, in which an area that consists of a large number of Wi-Fi AP is forming a single network. For example, the public Wi-Fi AP in urban areas, university campuses, business buildings, and industrial plants can form large-scope wireless local area networks (WLANs). Certainly, such networks generate a large amount of data continuously from the connected mobile devices. With the network virtualization technologies, data requesters within the WLAN can acquire data from mobile devices in high speed. Further, with the standards such as Internet Engineering Task Force (IETF) multicast DNS (RFC 6762) and DNS-Based Service Discovery (RFC 6763), which has been widely applied in today's dynamic service discovery within WLAN, nodes can discover and communicate with one another without maintaining static IP addresses.

Besides the Wi-Fi AP-based WLAN, the small cell facilities, which are used in telecommunication to assist the larger cellular network, can also apply network virtualization technologies such as European Telecommunications Standards Institute (ETSI) Network Functions Visualization (NFV) (Mijumbi et al. 2016) to support high-speed data acquisition within the edge networks.

Mobile Edge

The mobile edge refers to the coverage of telecommunication providers' base transceiver stations (BTS). Recent research efforts in mobile computing utilize the BTS colocated computational resources to improve the data delivery performance (Guo et al. 2017). For example, Internet service providers can improve multimedia stream data delivery to mobile devices by utilizing BTS-associated servers. Further, applying NFV together with

ETSI - Mobile/Multi-access Edge Computing (MEC) standard, which enables BTS servers to provide virtual machines for computing and networking, many mobile ubiquitous services and applications can enhance the speed of their data transmission by programming customised routing paths on MEC servers. Compared to PMP2P and WLAN, mobile edge provides wider coverage. Hence it is more feasible for city-wide or intercity-based Mobile Big Data acquisition. WLAN is suitable for inner-city-based data acquisition, and PMP2P is useful in an infrastructure-less environment such as an area after natural disaster or crisis (Gardner-Stephen et al. 2013).

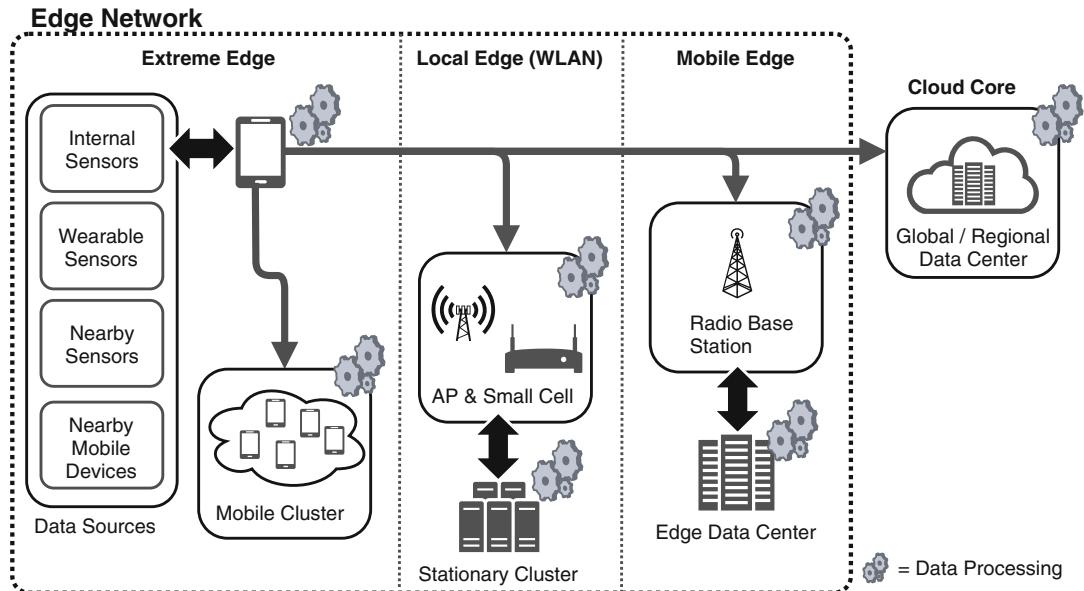
Data Processing

Recently, Mobile Big Data processing has two trends – the common global cloud-based processing and the edge-based data processing (Fig. 2).

Cloud-Centric Data Processing

As the amount of mobile data collected increased to the orders of tera- and petabytes, processing this huge data to extract relevant information became a complicated task. The first solutions looked at collecting the data at the cloud, as it provided unlimited data storage capabilities and processing power. On the cloud, distributed data processing approaches such as *MapReduce* were applied, producing relevant results. Earliest of these mobile data processing tasks included sensor data analytics and mining relevant to different domains such as healthcare, location-based services, etc. (Paniagua et al. 2012). Most of these tasks were batch-oriented and were ideal for cloud-centric data processing.

However, the next generation of mobile applications mainly dealt with streaming data and expected sub-second responses. The scenarios included processing stream data from multiple mobiles or data derived continuously from different regions, to generate novel results (Yang et al. 2013). To cope with these, streaming data analytic frameworks such as *Apache Flink*, *Apache Spark Streaming*, and *Apache Storm* have emerged. However, these solutions have introduced significant network latencies and



Mobile Big Data: Foundations, State of the Art, and Future Directions, Fig. 2 Data processing at the edge network comparing to cloud core. In order to achieve

performance needs, Mobile Big Data process can be distributed in different parts of edge network depending on the requirements of applications

failed to meet the application deadline demands. This has led to the edge analytics, where partial data processing such as error detection, filtering, and data consolidation have already happened at the edge devices. Further data analytic tasks requiring huge computational power still remained cloud-centric.

Edge-Based Data Processing

Recently, the needs of high-speed data processing for Internet of Things (IoT) systems, cyber-physical systems (CPS), and various mobile ubiquitous systems have motivated the fog computing architectures, which distribute the data processing tasks to computational resources located at different parts of edge networks (Chang et al. 2017). Similarly, edge analytics, which originally illustrate the importances of analyzing the big data derived from devices at the edge networks, has now involved distributing the edge analytic tasks to certain devices at the edge network in order to enhance the overall performance (IBM 2017). Further, edge-based data processing is also a major research trend in optimizing the performance of mobile cloud computing (Chen et al.

2016). To enumerate the recent models of edge-based data processing, we classified them in three types – extreme edge (i.e., PMP2P), local edge (i.e., WLAN), and mobile edge – corresponding to the three edge network types described in the previous section.

Extreme Edge

Extreme edge represents the devices located within WLAN or mobile edge (excluding the gateways), and they are capable of forming PMP2P. In mobile distributed computing domain, the devices of extreme edge are not only collecting or generating data, but they can also participate in data processing. Generally, depending on the environment and available resources, a system may distribute data processing in extreme edge network in two different models.

- **Sole Distribution.** Data stream, in cases that require rapid response, may push partial process to the mobile devices. Partitioning tasks to mobile devices, which is also the data source, is a common approach in mobile

cloud computing (Chang et al. 2016) and fog computing (Liyanage et al. 2016). Notably, dynamically allocating data process tasks among cloud and mobile nodes can improve the overall performance. For example, in a mobile sensing scenario where the mobile device is continuously collecting sensory data from its surrounding environment, the cloud-centric system can assign partial processes on the mobile device. Hence, it enables the corresponding mobile application to generate rapid results to the device user. Moreover, since the system distributes partial processes to mobile nodes, it reduces the burden of the central cloud server toward improving the throughput of the central server to serve more mobile nodes.

- **Group Distribution.** Today, in many crowded areas, people are surrounded with various mobile devices that have powerful computational, sensing, and storage capabilities. These devices are not only capable of providing data, they are also capable of forming cluster computing environments in an ad hoc manner. Specifically, architectures such as opportunistic computing (Conti et al. 2010), mobile edge cloud, or mobile ad hoc cloud computing (Chang et al. 2016) intend to introduce adaptive methods to enable Internet-less Mobile Big Data processing using proximity-based mobile ad hoc network resources. Although such an architecture faces challenges in incentive, reliability, and many mobility-related issues, it still shows a promising solution for situations that do not have accessibility to the Internet-based services. For example, the natural disaster may cause the network infrastructure corrupted. In such situation, the ad hoc-based network becomes a solution to swiftly establish an edge-based computing environment to collect and to process the data from all the participated mobile nodes.

Local Edge

The emerging fog computing architecture introduced by Cisco's research (Bonomi et al. 2012) was utilizing Internet gateway devices

(e.g., Cisco IR829 Industrial Integrated Router) to provide the similar model as utility cloud services in which the gateway devices provide virtualization technologies that allow system administrators to deploy software on them dynamically. Here, local edge represents Wi-Fi AP networks and small cells that cover various fog-compliant gateway devices. As mentioned previously, WLAN environments are commonly involved in large data transmission. Specifically, in IoT era, much data transmission in WLAN is derived from the systems that need to deliver the data to cloud for processing. Therefore, distributing such processes to local edge can enhance the overall performance and efficiency.

Local edge-based fog computing covers broad range of devices in WLAN. Essentially, the earlier proposal (Bonomi et al. 2012) illustrates the fog servers as the Internet routers or switches; recent research projects also experimented the feasibility of moving the data processes to the secondary gateway devices such as the gateway devices used to connect the non-Internet protocol (IP) network IoT devices (e.g., Bluetooth, IEEE 802.15.4, or Z-Wave devices) to IP network. Although these devices may not have ideal processing power and they do not involve large data transmissions, recent research project (Hajji and Tso 2016) has shown that adaptively utilizing these constraint gateway devices can still provide significant enhancement for the overall data processing. For example, an industrial plant may contain a large number of constraint gateway devices. If the system establishes a cluster computing environment using the constraint gateway devices, they can still provide a feasible computational resource to the mobile application nodes within the WLAN.

M

Mobile Edge

As mentioned previously, mobile edge denotes the BTS-based cellular mobile network environments. Since a large volume of mobile data derives from mobile Internet, the BTS networks are explicitly involving in large data transmission. Therefore, it motivates the MEC architecture in which the BTS nodes can integrate with virtualization technologies to provide edge-based utility

cloud services for computing and customized networking purposes to IoT/CPS and mobile ubiquitous applications.

MEC servers provide similar mechanism as fog servers in local edge. The major differences are the availability and coverage. Fundamentally, MEC servers have higher availability than local edge because MEC servers are within the cellular network infrastructure. On the other hand, local edge networks belong to specific parties such as university campus who have limited infrastructures.

Road Map: Challenges and Future Prospectives

Today, mobile data is not only large and massive, but it is also composed of various data types, including stream data and more. For these reasons, the challenges and future perspectives of innovation in the field of Mobile Big Data reside in its characteristics.

Volume

One of the characteristics of Mobile Big Data is the large volume of data. Till now, the increase of amount of data is managed by adding more storage (physical or online). For example, CDR data is in continuous increase, and there are many analytic applications performed based on them. Nevertheless, some types of information or attributes can be obsolete or unnecessary with proportion to their existence in time or to the analytic inputs. Therefore, by filtering or introducing some reduction methods such as preprocessing, graph theory, compression, redundancy elimination, and data mining, we can reduce some space (ur Rehman et al. 2016). However, we need to take into account the decompression overheads introduced to improve the efficiency. After all the volume still is not the most challenging aspect in Mobile Big Data, it is more on variety and velocity.

Velocity

The speed of executing and exchanging information is the most important factor in any ap-

plication. Since, the real-time information can make a huge difference in decision-making, it can affect the existence of markets, companies, industries, etc. Hence, the straightforward solution is to increase the central server capabilities by introducing load balancing techniques and optimization in defining a threshold to balance the load among data centers based on heterogeneity of data centers and also by minimizing the remote communication among them in order to improve the network infrastructure (Samadi and Zbakh 2017). An alternate solution is to reduce the data stream by applying data preprocessing. For example, in Neumeyer et al. (2010), the authors point to the challenges and the need for sophisticated approaches to apply real-time processing. From this perspective, the question is at which level we shall process the data, is it at the device level, the network level, or the server level. In some articles it was suggested to handle the streams at their first arrival and apply incremental techniques (Krempl et al. 2014) or by applying heterogeneity processing on the data (ur Rehman et al. 2017).

Variety

Mobile Big Data can take many forms such as messages, logs, records, calls, signatures, signals, images, streams, etc. In addition, the strong involvement and role of mobile devices in our lives make them a source of enormous streams of data associated to peoples activities, relationships, and locations. Hence, the diversity of types of data are collected today (structured, semi-structured, unstructured). As a result, the challenges can be resumed to two main aspects: maintaining the quality of the data providers and standardizing the data format. The quality check of the data can be maintained by introducing adaptive middleware (Schantz et al. 2003) that can help in preserving the quality of service and the data via appropriate resource management binding and scheduling services and policies. Another solution is by integrating data quality lifecycle into the architecture design as illustrated in Scannapieco et al. (2004). Concerning the challenge of standardizing the format of data, it can be inspired from other concepts such as IoT-Lite

(Bermudez-Edo et al. 2017) by pioneering a data specification for data format.

Variability

The variability is about the stream data coming from different sources in different speeds and qualities. Hence, we have to face the challenge of handling the elasticity of resources in the need to connect, match, clean, and transform (Gandomi and Haider 2015). When it is about quality of the software data, it is manageable and can be controlled; however, when it is about the devices, it can be very heterogeneous since the user's behavior or the environment can easily influence the data (ur Rehman et al. 2017).

Veracity

The Mobile Big Data can contain data with inherited unreliability. The best example is social media (Zhang et al. 2016a), where we have to deal with emotional and opinion information, which can be very uncertain but important at the same time. This complexity is addressed simply by researching the use of analytic techniques (Gandomi and Haider 2015) in developing management of mining and detecting unreliable data and also by increasing security levels. Besides, Mobile Big Data involves the notion of trust in the interactions and sharing services or information such as mobile banking data (Lee and Chung 2009).

Value

The value of data depends on the outcome of data analytics on the data. Since, by using analytics methods on it, we get an idea about what the data can offer in terms of knowledge and insights, which leads to defining its utility and usefulness for making profits (LaValle et al. 2011). Moreover, the mobile data has a potential to generate considerable revenues using IoT infrastructure sensing as a service to solve many problems related to smart city concept, such as fleet management, transportation problems, healthcare, and public administration (Perera et al. 2014).

Conclusion

Emerging ubiquitous mobile services and applications have unveiled the Mobile Big Data era in which large volumes of data are derived from mobile Internet traffic. This data is ideal in supporting various personal and public services such as personal recommendation services, spatiotemporal event detection, social behavior analytics, network resource planning, and many more. This chapter discussed in detail the various sources that produced these large volumes of data and how the mobile data is managed in terms of data acquisition and processing. For data processing, approaches such as cloud-centric data processing and edge-based data processing are discussed in detail. While the state of the art of the Mobile Big Data is interesting and evolving, it still leaves significant scope for future work, which the chapter tried to address in terms of the six major characteristics of the big data – volume, velocity, variety, variability, veracity, and value.

M

Cross-References

- ▶ Applications of Big Spatial Data: Health
- ▶ Big Data in Mobile Networks
- ▶ Big Data in Network Anomaly Detection
- ▶ Big Data in Smart Cities
- ▶ Linked Geospatial Data
- ▶ Network-Level Support for Big Data Computing
- ▶ Orchestration Tools for Big Data
- ▶ Spatial Data Integration
- ▶ Spatial Data Mining
- ▶ Streaming Big Spatial Data
- ▶ User Mobility Planning

Acknowledgements The work is supported by the Estonian Centre of Excellence in IT (EXCITE), funded by the European Regional Development Fund.

References

- Algizawy E, Ogawa T, El-Mahdy A (2017) Real-time large-scale map matching using mobile phone data. ACM Trans Knowl Discov Data (TKDD) 11(4):52

- Astarita V, Giofrè VP, Vitale A (2016) A cooperative intelligent transportation system for traffic light regulation based on mobile devices as floating car data (FCD). *Am Sci Res J Eng Technol Sci (ASRJETS)* 19(1): 166–177
- Bellairs J, Hlozek J, Egan T, Kuttel M (2016) An eHealth android application for mobile analysis of microplate assays. In: 2016 IST-Africa week conference. IEEE, pp 1–8
- Bermudez-Edo M, Elsaleh T, Barnaghi P, Taylor K (2017) IoT-lite: a lightweight semantic model for the Internet of things and its use with dynamic semantics. *Pers Ubiquit Comput* 21(3):475–487
- Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on mobile cloud computing. ACM, pp 13–16
- Chang C, Srirama NS, Buyya R (2016) Mobile cloud business process management system for the internet of things: a survey. *ACM Comput Surv (CSUR)* 49(4):70
- Chang C, Srirama NS, Buyya R (2017) Indie fog: an efficient fog-computing infrastructure for the internet of things. *Computer* 50(9):92–98
- Chen X, Jiao L, Li W, Fu X (2016) Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans Netw* 24(5):2795–2808
- Cheng Y-C, Chawathe Y, LaMarca A, Krumm J (2005) Accuracy characterization for metropolitan-scale WiFi localization. In: Proceedings of the 3rd international conference on mobile systems, applications, and services. ACM, pp 233–245
- Cheng X, Fang L, Hong X, Yang L (2017) Exploiting mobile big data: sources, features, and applications. *IEEE Netw* 31(1):72–79
- Chittaranjan G, Blom J, Gatica-Perez D (2013) Mining large-scale smartphone data for personality studies. *Pers Ubiquit Comput* 17(3):433–450
- Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 (2017). <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf>. Accessed 19 Oct 2017
- Conti M, Giordano S, May M, Passarella A (2010) From opportunistic networks to opportunistic computing. *IEEE Commun Mag* 48(9):126
- El Khaddar MA, Harroud H, Boulmaf M, Elkoutbi M, Habbani A (2012) Emerging wireless technologies in e-health trends, challenges, and framework design issues. In: 2012 international conference on multimedia computing and systems (ICMCS). IEEE, pp 440–445
- Ericsson Mobility Report. June 2017. Publisher: Niklas Heuveldop (2017). <https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf>. Accessed 19 Oct 2017
- Gandomi A, Haider M (2015) Beyond the hype: big data concepts, methods, and analytics. *Int J Inf Manage* 35(2):137–144
- Gardner-Stephen P, Challans R, Lakeman J, Bettison A, Gardner-Stephen D, Lloyd M (2013) The serval mesh: a platform for resilient communications in disaster & crisis. In: 2013 IEEE global humanitarian technology conference (GHTC). IEEE, pp 162–166
- Guo J, Song B, Yu RF, Yan Z, Yang TL (2017) Object detection among multimedia big data in the compressive measurement domain under mobile distributed architecture. *Futur Gener Comput Syst* 76:519–527
- Hadachi A, Batrashev O, Lind A, Singer G, Vainikko E (2014) Cell phone subscribers mobility prediction using enhanced Markov chain algorithm. In: 2014 IEEE intelligent vehicles symposium proceedings. IEEE, pp 1049–1054
- Hajji W, Tso PF (2016) Understanding the performance of low power raspberry pi cloud for big data. *Electronics* 5(2):29
- Herring R, Hofleitner A, Abbeel P, Bayen A (2010) Estimating arterial traffic conditions using sparse probe data. In: 2010 13th international IEEE conference on intelligent transportation systems (ITSC). IEEE, pp 929–936
- IBM (2017) Edge analytics cookbook. <https://developer.ibm.com/iotplatform/resources/edge-analytics-cookbook/>. Accessed 12 Oct 2017
- Jin X, Wah BW, Cheng X, Wang Y (2015) Significance and challenges of big data research. *Big Data Res* 2(2):59–64
- Krempel G, Źliobaite I, Brzeziński D, Hüllermeier E, Last M, Lemaire V, Noack T, Shaker A, Sievi S, Spiliopoulou M et al (2014) Open challenges for data stream mining research. *ACM SIGKDD Explor Newsl* 16(1):1–10
- Kwon L, Long K, Wan Y, Yu H, Cunningham B (2016) Medical diagnostics with mobile devices: comparison of intrinsic and extrinsic sensing. *Biotechnol Adv* 34(3):291–304
- LaValle S, Lesser E, Shockley R, Hopkins SM, Kruschwitz N (2011) Big data, analytics and the path from insights to value. *MIT Sloan Manag Rev* 52(2):21
- Lee CK, Chung N (2009) Understanding factors affecting trust in and satisfaction with mobile banking in Korea: a modified DeLone and McLean's model perspective. *Interact Comput* 21(5–6):385–392
- Lind A, Hadachi A, Batrashev O (2017a) A new approach for mobile positioning using the CDR data of cellular networks. In: 2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS). IEEE, pp 315–320
- Lind A, Hadachi A, Piksav P, Batrashev O (2017b) Spatio-temporal mobility analysis for community detection in the mobile networks using CDR data. In: 2017 9th international congress on ultra modern telecommunications and control systems (ICUMT)
- Liyanage M, Chang C, Srirama NS (2016) MePaaS: mobile-embedded platform as a service for distributing fog computing to edge nodes. In: 2016 17th international conference on parallel and distributed computing, applications and technologies (PDCAT). IEEE, pp 73–80
- Mijumbi R, Serrat J, Gorricho J-L, Bouting N, De Turck F, Boutaba R (2016) Network function virtualization:

- state-of-the-art and research challenges. *IEEE Commun Surv Tutorials* 18(1):236–262
- Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. Retrieved from <https://bitcoin.org/bitcoin.pdf>
- Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: 2010 IEEE international conference on data mining workshops (ICDMW). IEEE, pp 170–177
- Paniagua C, Flores H, Srirama NS (2012) Mobile sensor data classification for human activity recognition using MapReduce on cloud. *Proc Comput Sci* 10: 585–592
- Perera C, Zaslavsky A, Christen P, Georgakopoulos D (2014) Sensing as a service model for smart cities supported by internet of things. *Trans Emerg Telecommun Technol* 25(1):81–93
- Rathore MM, Ahmad A, Paul A, Rho S (2016) Urban planning and building smart cities based on the internet of things using big data analytics. *Comput Netw* 101:63–80
- Rueppel U, Stuebbe MK (2008) BIM-based indoor-emergency-navigation-system for complex buildings. *Tsinghua Sci Technol* 13:362–367
- Samadi Y, Zbakh M (2017) Threshold-based load balancing algorithm for big data on a cloud environment. In: Proceedings of the 2nd international conference on big data, cloud and applications. ACM, p 18
- Scannapieco M, Virgillito A, Marchetti C, Mecella M, Baldoni R (2004) The daquinis architecture: a platform for exchanging and improving data quality in cooperative information systems. *Inf Syst* 29(7): 551–582
- Schantz ER, Loyall PJ, Rodrigues C, Schmidt CD, Krishnamurthy Y, Pyarali I (2003) Flexible and adaptive QoS control for distributed real-time and embedded middleware. In: Proceedings of the ACM/IFIP/USENIX 2003 international conference on middleware. Springer, New York, pp 374–393
- Shen W-L, Chen C-S, Lin CK-J, Hua AK (2014) Autonomous mobile mesh networks. *IEEE Trans Mobile Comput* 13(2):364–376
- Sikder R, Uddin MJ, Halder S (2016) An efficient approach of identifying tourist by call detail record analysis. In: International workshop on computational intelligence (IWCI). IEEE, pp 136–141
- ur Rehman HM, Liew SC, Abbas A, Jayaraman PP, Wah YT, Khan US (2016) Big data reduction methods: a survey. *Data Sci Eng* 1(4):265–284. [Online]. Available <https://doi.org/10.1007/s41019-016-0022-0>
- ur Rehman HM, Liew SC, Wah YT, Khan KM (2017) Towards next-generation heterogeneous mobile data stream mining applications: opportunities, challenges, and future research directions. *J Netw Comput Appl* 79:1–24
- Wang K, Shao Y, Shu L, Zhu C, Zhang Y (2016) Mobile big data fault-tolerant processing for eHealth networks. *IEEE Netw* 30(1):36–42
- Yang L, Cao J, Yuan Y, Li T, Han A, Chan A (2013) A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Perform Eval Rev* 40(4): 23–32
- Zandbergen AP (2009) Accuracy of iPhone locations: a comparison of assisted GPS, WiFi and cellular positioning. *Trans GIS* 13(s1):5–25
- Zhang X, Yi Z, Yan Z, Min G, Wang W, Elmokashfi A, Maharjan S, Zhang Y (2016a) Social computing for mobile big data. *Computer* 49(9): 86–90
- Zhang M, Xu F, Li Y (2016b) Mobile traffic data decomposition for understanding human urban activities. In: 2016 IEEE 13th international conference on mobile ad hoc and sensor systems (MASS). IEEE, pp 1–9
-
- ## Moving Objects
- Spatiotemporal Data: Trajectories
-
- ## Multi-data Center Replication
- Geo-replication Models
-
- ## Multidimensional Process Analysis
- Multidimensional Process Analytics
-
- ## Multidimensional Process Analytics
- Multidimensional Process Analytics
-
- ## Synonyms and Related Terms
- Multidimensional process analysis; Multidimensional process mining; Process warehousing

Definitions

Multidimensional process analytics (MPA) augments business process analytics with the multidimensional perspective on the analysis data. The latter is typically event data that is produced during the execution of process instances. Classically, business process analytics “*is the family of methods and tools that can be applied to these events streams in order to support decision-making in organizations*” (zur Muehlen and Shapiro 2015). On top of just looking at the event streams, MPA enables their multidimensional representation and based on this the in-depth analysis of different process perspectives, e.g., activities and resources, as well as the comparison between different process execution variants or versions.

Overview

Generally, business process analytics (BPA) is concerned with the following tasks (zur Muehlen and Shapiro 2015):

- *Process Controlling* aims at the ex post analysis of event data stored from process instance executions.
- *Business Process Activity Monitoring* aims at monitoring process executions online, i.e., during execution, by providing information on, for example, critical events. Such information is presented in dashboards or by notifications such as alerts.
- *Process Intelligence* aims at finding unknown information from either real-world process event data or simulation data. Results can be suggestions for process optimization.

In lifting BPA to a multidimensional level, two main lines of existing work can be identified: (a) approaches routing in data warehousing (\mapsto *Process Warehousing (PWH)*) and (b) approaches routing in process mining/analysis (\mapsto *Multidimensional Process Mining (MPM)*). This is somehow obvious as BPA resides at the intersec-

tion of both worlds and a common challenge to both directions is the multidimensional modeling of process event data. However, differences in the analysis questions for PWH and MPM can be identified. Where PWH approaches reflect on the analysis of process data for different perspectives, MPM approaches do (also) consider the comparison of different process versions and variants, i.e., “*MPM does not discover a process model from a single event log, but multiple models from a set of sublogs*” (Vogelgesang et al. 2016a).

In summary, the key task of MPA is to model event data in a multidimensional fashion employing models and techniques from data warehousing in order to employ analysis techniques such as online analytical processing (OLAP) and data/process mining.

Key Research Findings

The first key task of MPA is the multidimensional modeling of process data. As common for the multidimensional analysis of data, e.g., in data warehouses (DWH), OLAP operations can then be applied to the multidimensional process database. The definition of the OLAP operations (second key task) depends then on the underlying data model. As the results of the multidimensional analysis can be complex, their adequate presentation constitutes the third key task. These three key tasks will be detailed in the following.

Multidimensional Data and Event Modeling

As motivated in section “[Overview](#),” there are two lines of data modeling, i.e., PWH inspired from data warehousing and MPM based on process logs. Figure 1a illustrates the basic idea of data modeling in PWH (top) and MPM (bottom) using a star schema. Star schemas along with snowflake schemas are the commonly used data models for relational data modeling in DWH (Chaudhuri and Dayal 1997). Roughly speaking, for both schemas, the analysis data is modeled as facts and stored in so-called fact tables. The facts can be analyzed along different dimensions that are represented at different granularity levels.

A commonly used example is dimension time with granularity levels day, month, and year.

For PWH, facts refer to performance measures of process elements, e.g., the cost of the process execution. Dimensions represent process perspectives that might influence the facts, most prominently time, resources, and behavior data.

Data modeling in MPM assumes a relation between cases (i.e., process executions) possessing attributes and being recorded on the basis of events. The attributes can be exploited as dimensions, for example, the age of a patient being classified into the age in years and into an age group at a more coarse-grained level. Also the events can be analyzed along dimensions, for example, the activity type. van der Aalst (2013) provides a multidimensional process cube; typical dimensions comprise time, case type, and event class. Figure 1b depicts an example of MPM data model (cf. Vogelgesang and Appelrath 2015).

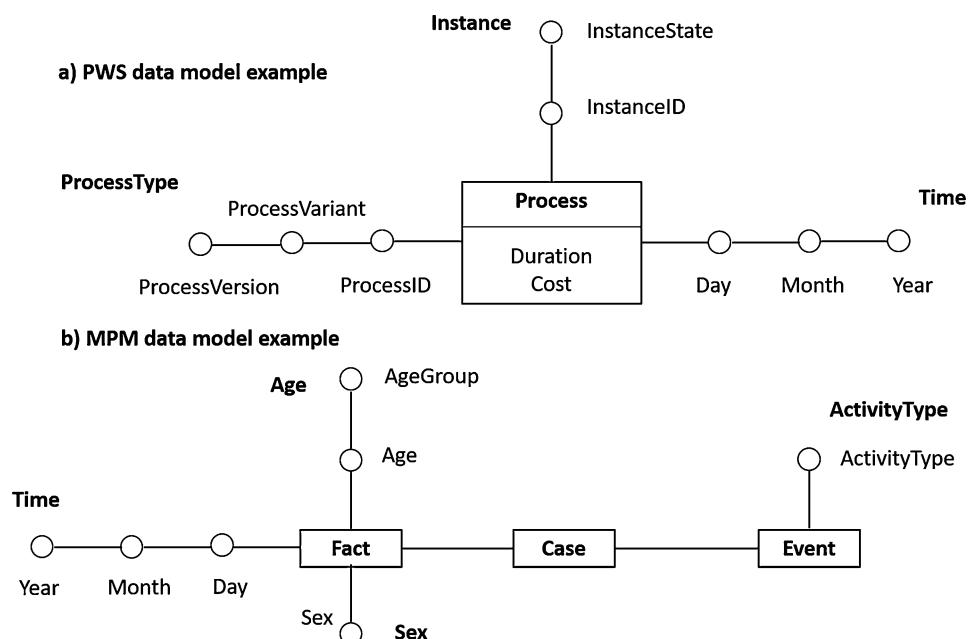
The differences in data modeling also lead to a difference in what the cells of the cubes for PWH and MPM contain. For PWH, the cells contain the numeric values for the facts such as cost,

duration, etc. For MPM the cells of the process cubes contain logs and the associated process models, respectively. In other words, from a log with a corresponding model, the combination of dimensions leads to a fragmentation of the logs into the cells. The example of a Dutch municipality in van der Aalst (2013) shows a sub log for a cell for a silver costumer (case type) for sales (event class) and for the year 2012.

Definition of OLAP Operations

Typically, OLAP operations operate along or across the dimensions of a data cube, for example, drilling down into the lower granularity or rolling up toward a higher granularity. Slicing and dicing enable the projection on dimensions and/or the selection of ranges of different dimensions.

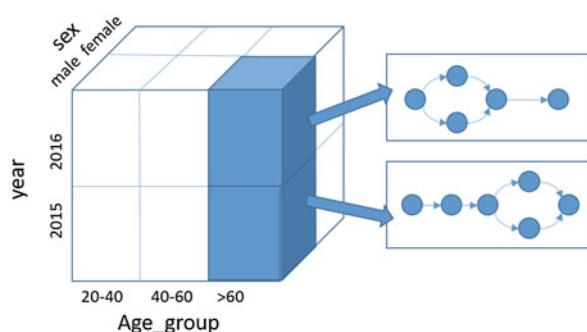
For PWH, in principle, the traditional OLAP operations can be applied as the logic of the cells and dimensions of PWH corresponds to the logic of DWH data cubes. Specifically, aggregation operations such as summing up or calculating the average over facts is possible if the facts refer to numeric facts related to business process



Multidimensional Process Analytics, Fig. 1 (a) Multidimensional data modeling in PWH, cf. Grossmann and Rinderle-Ma (2015) and (b) in MPM, cf. Vogelgesang and Appelrath (2015)

a) PWH view

Avg_duration		Chemo	Diabetes	AVG
2015	Running	8	10	9
	Completed	7	9	8
	AVG	7,5	9,5	8,5
2016	Running	8	9	8,5
	Completed	6	10	8
	AVG	7	9,5	8,25
AVG		7,25	9,5	8,375

b) MPM view

Multidimensional Process Analytics, Fig. 2 Results of applying OLAP operations on (a) PWH and (b) MPM. (The latter adapted from Cordes et al. 2014)

performance indicators, e.g., throughput. For the PWH schema in Fig. 1b, for example, one could roll up the average process duration to instance state (dimension instance) to year (dimension time) and to process versions (dimension process type). The view resulting from this roll up is depicted in Fig. 2a as a table structure. For instance, the average duration for completed instances of process-type diabetes was 9 in 2015.

Applying the OLAP operations in a traditional manner, however, is not possible for MPM as the cells of the process cubes do not contain facts as numeric values but logs and process models. Hence, a key finding in research is to define OLAP operations on logs and process models with specific challenge on finding appropriate aggregation functions (obviously, summing up logs and models is not possible).

Such OLAP operations for MPM are provided in Vogelgesang and Appelrath (2015). Here the user formulates OLAP queries which are translated into generic SQL queries that select event log attributes based on joins of the facts over the specified dimensions with filter conditions. It is also possible to apply aggregation on event basis based on the time stamps of the events.

Similarly, van der Aalst (2013) introduces the basis for OLAP operations on process cubes based on splitting or merging the logs in the cells. As a rule of thumb “[t]he case type and time window dimensions [...] are often used to merge or split a log vertically. The event class dimension is often used to merge or split a log horizontally”

(van der Aalst 2013). Splitting and merging serve as basis for defining OLAP operations roll up and drill down as well as slice and dice.

The results of applying OLAP operations are logs that can be mined for process models using existing process mining techniques (cf. van der Aalst 2016). Figure 2b shows the result of rolling up the process cube based on the schema in Fig. 1b. The highlighted cells contain the log data and the resulting models of the treatment processes for male patients beyond 60 for the years 2015 and 2016.

Visualization of Results

Figure 2 already indicates the different ways the results of multidimensional process analysis can be presented to analysts. While for PWH the results of OLAP operations can be conveyed based on, e.g., pivot tables or dashboards, the results of MPM are process models that can be displayed in a graphical way as typical for process mining techniques (e.g., as Petri nets or heuristic nets). What is more difficult is the presentation of differences between cells in the process cubes. However, such differences are usually of high interest to analysts to, for example, compare developments over time. For example, one would like to compare the treatment process executions for male patients beyond 60 years for 2016 with the corresponding model for 2015. Have there been any differences in how this patient group was treated? Contrary to comparing the values of

two cells as for PWH, comparing process models for differences is more difficult. There are general approaches for calculating differences between process models, e.g., Kriglstein et al. (2013), and approaches specific for comparing the results of MPM, cf. Cordes et al. (2014). In any case, the differences between the models are displayed using a difference graph that holds one of the models as basic version and depicts the difference to the other model using colors.

Examples of Application

Several applications of MPM to real-world event logs from different domains can be found. A selection comprises

Health Care

Hospital data was analyzed with PMCube (Vogelgesang and Appelrath 2015): several cubes with different combinations of dimensions were formed based on OLAP operations, for example, medical department and urgency or age and sex. In Vogelgesang et al. (2016a), the approach was combined with change logs (cf. e.g., Kaes and Rinderle-Ma 2015) in order to explore exceptional cases and situations where several of those were found, e.g., the “termination of anesthesia was initiated before the clearance of the surgeon” (Vogelgesang et al. 2016a).

Higher Education

Two case studies on MPM in higher education are available, i.e., applying PMCube on the HEP (higher education processes) data set (Vogelgesang et al. 2016a) and the Process Cubes approach on video lecture data from Eindhoven University of Technology (Bolt et al. 2015). The first case study data shows, for example, differences in conducting course processes when compared two 2 different years. In later courses, less exercises but more tutorials were provided. The second case study aims at defining analytic workflows for generating reports and statements about the lectures such as that student performance increases with watching the video. The

created reports and statements were evaluated with domain experts, i.e., lecturers.

Another case study on process data from Dutch Municipalities can be found in van der Aalst (2013).

Future Directions for Research

Although the approaches for PWH date back to the late 1990s, the approaches on MPM have just recently been developed, in the course of the advent and development of process mining. So the discipline of multidimensional process analytics still bears many open research questions which are shortly discussed in the following (based on van der Aalst (2013) and Vogelgesang et al. 2016a). The challenges hold specifically for MPM, and their coverage by existing literature is discussed in Vogelgesang et al. (2016a).

M

Performance

In general, OLAP operations on multidimensional data can become – depending on the data model – quite complex, e.g., if many join operations are required. This observation is also valid for OLAP operations on MPM data, where the operations might become even more complex as the underlying data, event logs, is more complex as for DWH and PWH data. Hence, the performance of conducting the OLAP operations plays a crucial role. OLAP query optimization in MPM is addressed by PMCube (Vogelgesang and Appelrath 2015). PMCube transforms OLAP queries into SQL patterns and hence pushes aggregation and filtering steps to the database. The subsequent comparison of several prototypical implementations for MPM on datasets of different complexity PMCs shows the best performance. Nonetheless, research on how to run OLAP operations for MPM efficiently can be extended in several directions, e.g., by looking at techniques from DWH research such as investigating query optimization, parallel processing, and index structures and using materialized views (Chaudhuri and Dayal 1997).

Presentation of Results and Interactivity

There are some approaches on visualizing the results of comparing cells in process cubes (cf. section “[Visualization of Results](#)”). Still the question remains on how to visualize differences between a higher number of cells, i.e., several process models as most existing approaches are restricted to visualizing the differences between two models, e.g., Kriglstein et al. (2013). The other research direction is to improve the interactivity of conducting multidimensional analysis on process data which is a crucial brick in explorative analysis. Interactivity mainly refers to the interaction with analysis results, i.e., process models. This mixes in with research on process visualization and process mining where interactivity has been identified as general challenge (van der Aalst et al. 2011). So the topic splits into providing general ways of interacting with process models but has specific challenges for MPA as the underlying data is calculated in a potentially complex manner and it hence can become expensive to recalculate results. Thus there is also an intersection with the challenge on performance (cf. section “[Performance](#)”). First approaches specific to MPM have been developed. Vogelgesang et al. (2016b), for example, introduce the concept of operation stacks in order to enable undo and redo of analysis steps. Research on interactivity is nonetheless open for extended interaction with process models and interaction in the context of the comparison of cells.

Changes in Data

Due to constant changes in applications and their environments, data is likely to be changed, i.e., new data is created, deleted, or updated. In order to keep analysis results up-to-date, the changes in the data have to be considered in the MPA. This issues has been researched well in DWH, specifically treated for updating materialized views (Chaudhuri and Dayal 1997). The core question here is how to update views by propagating the updates incrementally, instead of recalculating the views. As business process is constantly subject to change as well, the problem directly translates to MPA. For PWH being logically close to DWH, existing techniques for view updates

are likely to be applicable (although no specific statements on this were found). For MPM, where the problem of data change is referred to as concept drift (van der Aalst et al. 2011), updating materialized views is more challenging as the data and the OLAP operations to build the views are already more complex. Note that changes might not only affect the data within process cells but also attributes that form the dimensions such as weather data (van der Aalst 2013).

Conclusion

This list of future research directions is not complete and might also evolve with new insights into existing challenges. MPA is a relatively new research topic with ties to the more established research areas DWH and PWH. With new developments such as huge data volumes, streaming data, and distributed analysis of data, the above mentioned challenge will aggravate, and new challenges appear.

Cross-References

- ▶ [Business Process Event Logs and Visualization](#)
- ▶ [Event Log Cleaning for Business Process Analytics](#)

References

- Bolt A, de Leoni M, van der Aalst WMP, Gorissen P (2015) Exploiting process cubes, analytic workflows and process mining for business process reporting: a case study in education. In: Proceedings of the 5th international symposium on data-driven process discovery and analysis (SIMPDA 2015), Vienna, 9–11 Dec 2015, pp 33–47. <http://ceur-ws.org/Vol-1527/paper3.pdf>
- Chaudhuri S, Dayal U (1997) An overview of data warehousing and OLAP technology. SIGMOD Rec 26(1):65–74. <http://doi.acm.org/10.1145/248603.248616>
- Cordes C, Vogelgesang T, Appelrath H (2014) A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In: BPM 2014 international workshops, Eindhoven, 7–8 Sept 2014, Revised papers, pp 383–394. https://doi.org/10.1007/978-3-319-15895-2_32
- Grossmann W, Rinderle-Ma S (2015) Fundamentals of business intelligence. Data-centric systems and appli-

- cations. Springer. <https://doi.org/10.1007/978-3-662-46531-8>
- Kaes G, Rinderle-Ma S (2015) Mining and querying process change information based on change trees. In: Proceedings of 13th international conference, ICSOC 2015, Goa, 16–19 Nov 2015, pp 269–284. https://doi.org/10.1007/978-3-662-48616-0_17
- Kriglstein S, Wallner G, Rinderle-Ma S (2013) A visualization approach for difference analysis of process models and instance traffic. In: Proceedings of 11th international conference, BPM 2013, Beijing, 26–30 Aug 2013, pp 219–226. https://doi.org/10.1007/978-3-642-40176-3_18
- van der Aalst WMP (2013) Process cubes: slicing, dicing, rolling up and drilling down event data for process mining. In: First Asia pacific conference, AP-BPM 2013, Beijing, 29–30 Aug 2013. Selected papers, pp 1–22. https://doi.org/10.1007/978-3-319-02922-1_1
- van der Aalst WMP (2016) Process mining – data science in action, 2nd edn. Springer. <https://doi.org/10.1007/978-3-662-49851-4>
- van der Aalst WMP et al (2011) Process mining manifesto. In: BPM 2011 international workshops, Clermont-Ferrand, 29 Aug 2011, Revised selected papers, Part I, pp 169–194. https://doi.org/10.1007/978-3-642-28108-2_19
- Vogelgesang T, Appelrath H (2015) A relational data warehouse for multidimensional process mining. In: Proceedings of the 5th international symposium on data-driven process discovery and analysis (SIMPDA 2015), Vienna, 9–11 Dec 2015, pp 64–78. <http://ceur-ws.org/Vol-1527/paper5.pdf>
- Vogelgesang T, Kaes G, Rinderle-Ma S, Appelrath H (2016a) Multidimensional process mining: questions, requirements, and limitations. In: Proceedings of the CAiSE'16 forum, at the 28th international conference on advanced information systems engineering (CAiSE 2016), Ljubljana, 13–17 June 2016, pp 169–176. <http://ceur-ws.org/Vol-1612/paper22.pdf>
- Vogelgesang T, Rinderle-Ma S, Appelrath H (2016b) A framework for interactive multidimensional process mining. In: Business process management workshops – BPM 2016 international workshops, Rio de Janeiro, 19 Sept 2016, Revised papers, pp 23–35. https://doi.org/10.1007/978-3-319-58457-7_2
- zur Muehlen M, Shapiro R (2015) Business process analytics. In: Handbook on business process management 2, strategic alignment, governance, people and culture, 2nd edn, pp 243–263. https://doi.org/10.1007/978-3-642-45103-4_10

M

Multidimensional Process Mining

- ▶ Multidimensional Process Analytics

Multi-instance Process Mining

- ▶ Artifact-Centric Process Mining

N

Native Distributed RDF Systems

Marcin Wylot¹ and Sherif Sakr²

¹ODS, TU Berlin, Berlin, Germany

²Institute of Computer Science, University of Tartu, Tartu, Estonia

Synonyms

Distributed SPARQL query processing; Scalable SPARQL query processors

Definitions

RDF (<https://www.w3.org/RDF/>), the Resource Description Framework, represents a main ingredient and data representation format for Linked Data and the Semantic Web. It supports a generic graph-based data model and data representation format for describing things, including their relationships with other things. RDF is designed to flexibly model schema-free information which represents data objects as *triples* in the form (S, P, O) , where S represents a subject, P represents a predicate, and O represents an object. A triple indicates a relationship between S and O captured by P . Consequently, a collection of triples can be modeled as a directed graph where the graph vertices denote subjects and

objects, while graph edges are used to denote predicates. The SPARQL (<https://www.w3.org/TR/sparql11-overview/>) query language has been recommended by the W3C as the standard language for querying RDF data. A SPARQL query Q specifies a graph pattern P which is matched against an RDF graph G . The query matching process is performed via matching the variables in P with the elements of G such that the returned graph is contained in G (pattern matching). In practice, most RDF stores can be searched using queries that are composed of *triple patterns*. A triple pattern is much like a triple, except that S , P , and O can be replaced by variables. Similar to triples, triple patterns are modeled as directed graphs.

RDF has been gaining widespread momentum and usage in different domains such as the Semantic Web, Linked Data, Open Data, social networks, digital libraries, bioinformatics, or business intelligence. As an example of this trend, a growing number of ontologies and knowledge bases storing millions to billions of facts such as DBpedia, and Wikidata are now publicly available. With increasing sizes of RDF datasets, executing complex queries on a single node has turned to be impractical especially when the node's main memory is dwarfed by the volume of the dataset. Therefore, there was a crucial need for distributed systems with a high degree of parallelism that can satisfy the performance demands of complex SPARQL queries. Several distributed RDF processing systems have been introduced where the storage

and query processing of linked data are managed on multiple nodes. In general, distributed RDF systems are characterized by larger aggregate memory sizes and higher processing capacity. On the other hand, they might incur significant intermediate data shuffling when answering complex SPARQL queries that span multiple disjoint partitions. This entry gives an overview of various techniques and systems that have been *natively* designed and implemented from scratch to efficiently query large RDF datasets in distributed environments, i.e., native distributed RDF systems.

Overview

With increasing sizes of RDF datasets, executing complex queries on a single node has turned to be impractical especially when the node's main memory is dwarfed by the volume of the dataset. Therefore, there was a crucial need for distributed systems with a high-degree of parallelism that can satisfy the performance demands of complex SPARQL queries. Several distributed RDF processing systems have been introduced where the storage and query processing of linked data is managed on multiple nodes. Such distributed RDF systems are characterized by larger aggregate memory sizes and higher processing capacity. On the other hand, they might incur significant intermediate data shuffling when answering complex SPARQL queries that span multiple disjoint partitions. In this entry we give an overview of various techniques and systems for efficiently querying large RDF datasets in distributed environments.

Key Research Findings

To build an RDF data management system, the simplest approach is using relational databases to store the data, e.g., "*statement tables*" design pattern for storing RDF statements (Sakr and Al-Naymat 2010). From first few implementations, it shows that naive ways for using relational database as a black box show several shortcomings in terms of achieving the

performance and scalability of most use cases or datasets in reality. However, later generations of this kind significantly improve performance by designing more complicated table layouts associated with tailored indexing structures or even stored functions. Along with such strategies, alternative storage structures are proposed such as index permutations, property tables, vertical partitioning, graph-based storage, and binary storage (Sakr and Al-Naymat 2010). Among them, index permutations is pretty much the same with traditional statement tables but focuses on building faster access pattern by exhaustively indexing all possible patterns. The further modifications are exploiting data distribution of RDF graphs to partition the data in favor of collocation data items that retrieved together a query. Property tables and vertical partitioning are two approaches of this kind. Moving far away from traditional relational database design, graph-based storage and binary storage provide completely different storage structures with different focuses. The graph-based systems exploit the graph structure to build data structures that favor the graph exploration queries, e.g., subgraph matching. On the other hand, binary systems exploit the fast bitwise and compact bit/byte representations to be able to fit as much data into the main memory to achieve better performance and scale a bit better to certain dataset size. The single node systems remain limited by the computational power and memory capacities of a single machine. Therefore scaling the systems vertically in terms of adding a number of processing cores and adding more memory will soon hit its upper limits. To remedy such limitations of single node processing, the distributed processing infrastructure has been exploited. With the potential of a high degree of parallelism, a distributed system can satisfy the performance demands of complex SPARQL queries on very large RDF collections. The generic parallel processing platforms have their own shortcomings which motivated several systems to build their own underlying infrastructure to serve their requirements. The main research directions in these files focus on efficient partitioning RDF graph and distributed

query execution algorithms. Partitioning large graph data collections is a very challenging task; therefore many approaches have been developed to efficiently and effectively perform this operation. Some approaches suggest to partially duplicate the data in order to execute frequent queries entirely in parallel, without transferring intermediate results between nodes.

Examples of Systems

Several systems have been designed to handle RDF data and workloads in a distributed and scalable fashion. This section presents a selection of the most well-known approaches in this field. For example, Partout (Galárraga et al. 2014) is a distributed engine that relies on a workload-aware partitioning strategy for the RDF data by allowing queries to be executed over a minimum number of machines. Partout exploits a representative query load to collect information about frequently co-occurring sub-queries and for achieving optimized data partitioning and allocation of the data to multiple nodes. The architecture of Partout consists of a coordinator node and a cluster of n hosts that store the actual data. The coordinator node is responsible for distributing the RDF data among the host nodes, designing an efficient distributed query plan for a SPARQL query and initiating query evaluation. The coordinator does not have direct access to the actual data but instead utilizes global statistics of the RDF data, generated at partitioning time, for query planning. Each of the host nodes runs a centralized triple store, RDF-3X (Neumann and Weikum 2008). Queries are issued at the coordinator, which is responsible for generating a suitable query plan for the distributed query execution. The data is located at the hosts which are hosting the data partitions. Each host executes its part of the query over its local data and sends the results to the coordinator, which finally generates the query result. The global query optimization algorithm starts with a plan optimized with respect to the selectivity of the query predicates and then applies heuristics to obtain an efficient plan for the distributed setup. Each host relies on the

RDF-3X optimizer for optimizing its local query plan.

TriAD (Triple-Asynchronous-Distributed) (Gurajada et al. 2014) uses a main-memory and shared-nothing architecture which is based on an asynchronous message passing protocol. TriAD applies a classical master-slave architecture in which the slave nodes autonomously and asynchronously exchange messages to evaluate multiple join operations in parallel. Relying on asynchronous communication allows the sibling execution paths of a query plan to be processed in a freely multi-threaded fashion and only get merged (i.e., get synchronized) when the intermediate results of the entire execution paths are joined. TriAD employs six comprehensive combinations of indexing over the RDF elements. The indexes are maintained in a distributed main-memory data structure where each index is hash-partitioned according to its join key and locally sorted in lexicographic order. Therefore, TriAD can perform efficient, distributed merge joins over the hash-partitioned permutation lists. In addition, TriAD uses join-ahead pruning using an additional RDF summary graph. The join-ahead pruning is executed at the master node, in order to prune entire partitions of triples from the *SPO* lists that cannot contribute to the results of a given SPARQL query. TriAD uses a bottom-up dynamic programming mechanism for join-order enumeration and considers the locality of the index structures at the slave nodes, the data exchange cost of the intermediate results, and the option to execute sibling paths of the query plan in a multi-threaded fashion, in order to estimate the query execution plan with the cheapest cost.

The DREAM (Distributed RDF Engine with Adaptive Query Planner and Minimal Communication) system (Hammoud et al. 2015; Hasan et al. 2016) has been designed with the aim of avoiding partitioning RDF graphs and partitions SPARQL queries only, thus attempting to combine the advantages of the centralized and distributed RDF systems. DREAM stores a dataset intact at each cluster node and employs a query planner that partitions SPARQL queries. Specifically, the query planner transforms the query into a graph; decomposes this graph into sets of

subgraphs, each with a basic two-level tree structure; and maps each set to a separate machine. Afterward, all machines process their sets of subgraphs in parallel and coordinate with each other to return the final result. No intermediate data is shuffled and only minimal control messages and metadata are exchanged. To decide upon the number of sets (which dictates the number of machines) and their constituent subgraphs (i.e., the graph plan), the query planner enumerates various possibilities and selects a plan that results in the lowest network and disk costs for the graph. This is achieved through utilizing a cost model that relies on RDF graph statistics. Using the above approach, DREAM is able to select different numbers of machines for different query types, hence, rendering it adaptive.

Shi et al. (2016) present a distributed graph-based RDF store that leverages remote direct memory access (RDMA)-based graph exploration to provide highly concurrent and low-latency queries over large datasets. Shi et al. provide an RDMA-friendly distributed key/value store that provides differentiated encoding and fine-grained partitioning of graph data to reduce RDMA transfers. To support efficient scale-out, Shi et al. follow a graph-based design by storing RDF triples as a native graph and leverage graph exploration to handle queries. Shi et al. leverage full-history pruning to avoid the cost of expensive final join operations. Their system combines data migration for low latency and execution distribution for high throughput by leveraging the low latency and high throughput of one-sided RDMA operations. Consequently, it can avoid the costly centralized final join on the results aggregated from multiple machines. Shi et al. extend an RDF graph by introducing index vertices so that indexes are naturally parts of the graph. To partition and distribute data among multiple machines, they apply a differentiated partition schema to embrace both locality (for normal vertices) and parallelism (for index vertices) during query processing. Based on the observation that RDF queries only touch a small subset of graph data, the system further incorporates predicate-based finer-grained vertex decomposition and stores the decomposed graph

data into a refined, RDMA-friendly distributed hash table to reduce RDMA transfers. Depending on the selectivity and complexity of queries, the system decomposes a query into a sequence of sub-queries and handles multiple independent sub-queries simultaneously. For each sub-query, it adopts an RDMA communication-aware mechanism. For selective queries, it uses in-place execution that leverages one-sided RDMA read to fetch necessary data so that there is no need to move intermediate results. For nonselective queries, it uses one-sided RDMA WRITE to distribute the query processing to all related machines. To prevent large queries from blocking small queries when handling concurrent queries, Shi et al. provide a latency-centric work-stealing schema to dynamically oblige queries in straggling workers.

Cheng et al. (2015) present a hybrid method for processing RDF that combines similar-size and graph-based partitioning strategies. In this approach, similar-size partitioning is used for fast loading data, while graph-based partitioning is used to achieve efficient query processing. A two-tier index architecture is adopted. The first tier is a lightweight primary index which is used to maintain low loading times. The second tier is a series of dynamic, multilevel secondary indexes, computed during query execution, which is used to limit the inter-machine data transfer for subsequent operations for similar graph patterns. Additionally, this approach relies on a set of parallel mechanisms which combine the loading speed of similar-size partitioning with the execution speed of graph-based partitioning. For example, it uses fixed-length integer encoding for RDF terms and indexes that are based on hash tables to increase access speed. The indexing process does not use network communication in order to increase the loading speed. The local lightweight primary index is used to support very fast retrieval and avoid costly scans, while the secondary indexes are used to support nontrivial access patterns. The secondary indexes are built dynamically, as a by-product of query execution; thus they amortize costs of common expensive access patterns.

The DiploCloud system (Wylot et al. 2011; Wylot and Cudré-Mauroux 2016) has been

designed to use a hybrid storage structure co-locating semantically related data to minimize internode operations. The colocated data patterns are mined from both instance and schema levels. DiploCloud uses three main data structures: molecule clusters, template lists, and a molecule index. Molecule clusters extend property tables to form RDF subgraphs that group sets of related URIs in nested hash tables and to colocate data corresponding to a given resource. Template lists are used to store literals in lists, like in a columnar database system. Template lists allow to process long lists of literals efficiently; therefore, they are employed mainly for analytics and aggregate queries. The molecule index serves to index URIs based on the molecule cluster to which they belong. In the architecture of DiploCloud, the master node is composed of three main subcomponents: a key index encoding URIs into IDs, a partition manager, and a distributed query executor. The worker nodes of the system hold the partitioned data and its corresponding local indexes. The workers store three main data structures: a type index (grouping keys based on their types), local molecule clusters, and a molecule index. The worker nodes run sub-queries and send results to the master node. The data partitioner of DiploCloud relies on three molecule-based data partitioning techniques: (1) Scope-k Molecules that manually defines the size for all molecules, (2) manual partitioning where the system takes an input to manually define the shapes of molecules, (3) adaptive partitioning that starts with a default shape of molecules and adaptively increases or decreases the size of molecules based on the workload. Starlike queries are executed in parallel without any central coordination. For queries requiring distributed joins, DiploCloud picks one of the two execution strategies: (1) if the intermediate result set is small, DiploCloud ships everything to the master node that performs the join, and (2) if the intermediate result set is large, DiploCloud performs a distributed hash join.

The EAGRE (Zhang et al. 2013) system has been presented as an Entity-Aware Graph compREssion technique to encode RDF datasets

using key-value storage structures that preserve the structure and semantic information of RDF graphs. The main idea of this technique is to extract entities and entity classes from the original RDF in order to build a compressed RDF entity graph. The system adopts a graph partitioning mechanism that distributes RDF data across the worker nodes and designs an in-memory index structure to efficiently accelerate the evaluation of range and order-sensitive queries. In particular, the entity classes are partitioned on the computing nodes in a way that they preserve the structure locality of the original RDF graph. The evaluation process of a SPARQL query starts by identifying the entity classes using an in-memory index for the compressed RDF entity graph on a query engine. Then, the query is submitted to the worker nodes, which maintain RDF data, where the query coordinator on each worker participates in a voting process that decides the scheduling function of distributed I/Os. EAGRE uses a distributed I/O scheduling mechanism to reduce the cost of the disk scans and the total time for the query evaluation process. In practice, whenever some workers complete their local I/O operation, they exploit the scheduler to feed other workers with the gathered statistical information of the processed data.

SemStore (Wu et al. 2014) consists of the following main components: a data partitioner, a master node, and a number of computing nodes. The SemStore partitioner adopts a partitioning mechanism, Rooted Sub-Graph (RSG), that is designed to effectively localize all the queries in the shapes of a star, a chain, a tree, or a cycle that captures the most frequent SPARQL queries. The SemStore partitioner uses a k-means partitioning algorithm for assigning the highly correlated RSGs into the same machine. Each computing node builds local data indexes and statistics for its assigned subgraph and utilizes this information during local join processing and optimization. In addition, the data partitioner builds a global bitmap index over the vertices of the RDF graph and collects the global statistics. The master node is the SemStore component that receives the user query, builds the distributed query plan, and coordinates distributed data transfer between

the computing nodes. The computing nodes use TripleBit (Yuan et al. 2013) for local query evaluation.

Blazegraph (<https://www.blazegraph.com>) maintains three RDF indexes (SPA, POS, OSP) and leverages a B+Tree implementation. These indexes are dynamically partitioned into key-range shards that can be distributed between nodes in a cluster. Its scale-out architecture is based on multiple services. The shard locator service maps each key-range partition to a metadata record that allows to locate the partition. A transaction service coordinates locks to provide the isolation. A client service allows to execute distributed tasks. The query execution process starts with the translation of a SPARQL query to an abstract syntax tree. Then, the tree is rewritten to optimize the execution. Finally, it is translated to a physical query plan, vectorized, and submitted for execution.

Peng et al. (2016) propose a graph-based approach to evaluate SPARQL queries over a distributed RDF dataset. They adopt partial evaluation and assembly approach described by Jones et al. (1996). The method assumes that RDF data is partitioned among multiple machines in a cluster and it divides the query execution into two steps. First, each node partially evaluates the query with the data that it owns and builds a collection of intermediate results, i.e., partial matches. Second, the partial matches are joined either in centralized or distributed manner. To compute the partial matches, the authors employ their previous work on a centralized RDF query engine, gStore (Zou et al. 2014). The method builds the results step-by-step until no more matches can be found or the entire query is satisfied. Since the data is fragmented among many machines, the algorithm terminates when the query is satisfied up to the limits of the local data. In order to combine the partial matches, the authors first propose a centralized algorithm where all intermediate results are sent to one machine that performs an iterative join process. The second algorithm proposed by Peng et al. combines the partial matches in a distributed manner. By using bulk synchronous parallel model (Valiant 1990).

Chameleon-db (Aluc et al. 2013) has been proposed as a workload-aware RDF data management system that automatically and periodically adjusts its layout of the RDF database with the aim of optimizing the query execution time and auto-tuning its performance. Such adjustment is done in a way that enables partitions to be concurrently updated without the need of stopping the query processing. Similar to the previous approach, RDF data and SPARQL queries are represented as graphs, and queries are evaluated using a subgraph matching algorithm. However, the previous approach evaluates the queries over the entire RDF graph; chameleon-db partitions the RDF graph and prunes out the irrelevant partitions during query evaluation by using partition indexes. The main goal of the chameleon-db partitioning strategy is to carefully identify the graph partitions that truly contribute to the final results in order to minimize the number of irrelevant intermediate results for final joins. To prune the irrelevant partitions, chameleon-db uses an incremental indexing technique that uses a decision tree to keep track of which segments are relevant to which queries.

Wang et al. (2016) built on the system described by Peng et al. (2016) and proposed a distributed method to evaluate regular path queries (<https://www.w3.org/TR/sparql11-property-paths/>), i.e., queries over an RDF graph that retrieve resources connected by a sequence of predicates. The authors use partial evaluation to parallelize the execution within multiple nodes containing partitioned RDF triples. First, every node finds the property paths within its local data. Second, every node finds the possible connection with the previously computed paths and triples distributed among other machines. In order to compute the possible connections, the presented system maintains two special indexes pointing to directly connect resources on other machines. Then, all partial answers are grouped in parallel based on three types of partial answer. Finally, all grouped partial answers are sent to a coordinating node that performs joins to generate the final results.

AdHash (Harbi et al. 2015; Al-Harbi et al. 2016) is another distributed in-memory RDF

engine which initially applies lightweight hash partitioning that distributes the RDF triples by hashing on their subjects. AdHash attempts to improve the query execution times by increasing the number of join operations that can be executed in parallel without data communication through utilizing hash-based locality. In particular, the join patterns on subjects included in a query can be processed in parallel. The locality-aware query optimizer exploits this property to build a query evaluation plan that reduces the size of intermediate results transferred among the worker nodes. In addition, AdHash continuously monitors the data access patterns of the executed workload and dynamically adapts to the query workload by incrementally redistributing and replicating the frequently partitions of the graphs. The main goal for the adaptive dynamic strategy of AdHash is to effectively minimize or eliminate the data communication cost for future queries. Therefore, hot patterns are redistributed and potentially replicated to allow future workloads which contain them to be evaluated in parallel by all worker nodes without any data transfer. To efficiently manage the replication process, AdHash specifies a budget constraint and uses an eviction policy for the redistributed patterns. As a result, AdHash attempts to overcome the disadvantages of static partitioning schemes and dynamically reacts with changing workloads. The locality-aware query planner of AdHash uses the global statistics and the pattern index to decide if a query, in whole or partially, can be processed without communication. Queries that can be fully answered without communication are planned and executed by each worker independently. For queries that require communication, the planner exploits the hash-based data locality and the query structure to find a plan that minimizes communication and the number of distributed joins.

Potter et al. (2016) propose a distributed algorithm to answer queries over a partitioned RDF graph. The algorithm dynamically decides when and how to exchange data between the machines, i.e., it exchange data only when it is needed. The algorithm adapts the index-nested loop join, and at each stage it decides if the next stage should

be further executed on the same machine or if it should be moved/forked to another server. Specifically, after reordering the triple patterns from the query, each server tries to recursively find RDF triples which match the pattern. Each stage of the nested join loop, which corresponds to a triple pattern, takes as an input a triple pattern and the partial answers of the previous stages. Before evaluating a triple pattern with bindings from previous partial answers, the server verifies if there is another server in the cluster that has RDF triples which can contribute to the query execution. If there is a machine that can contribute to the query evaluation, it receives a message with the current partial answer, i.e., the answer including all previous stages of the query execution. The new machine can then continue the forked query evaluation with the data located on this machine. The presented variant of the index-nested loop join algorithm dynamically jumps to the relevant servers when it is required by the query execution process based on the query itself and the data partitioning. The servers exchange the messages asynchronously without static synchronization in the query plan.

N

References

- Al-Harbi R, Abdelaziz I, Kalnis P, Mamoulis N, Ebrahim Y, Sahli M (2016) Accelerating SPARQL queries by exploiting hash-based locality and adaptive partitioning. VLDB J 25(3):355–380. <https://doi.org/10.1007/s00778-016-0420-y>
- Aluc G, Ozu MT, Daudjee K, Hartig O (2013) Chameleon-db: a workload-aware robust RDF data management system. Technical report CS-2013-10, University of Waterloo
- Cheng L, Kotoulas S (2015) Scale-out processing of large RDF datasets. IEEE Trans Big Data 1(4):138–150. <https://doi.org/10.1109/TBDA.2015.2505719>
- Galárraga L, Hose K, Schenkel R (2014) Partout: a distributed engine for efficient RDF processing. In: 23rd international World Wide Web conference, WWW '14, Seoul, 7–11 Apr 2014. Companion volume, pp 267–268. <http://doi.acm.org/10.1145/2567948.2577302>
- Gurajada S, Seufert S, Miliaraki I, Theobald M (2014) TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing. In: International conference on management of data, SIGMOD 2014, Snowbird, 22–27 June 2014, pp 289–300. <http://doi.acm.org/10.1145/2588555.2610511>

- Hammoud M, Rabbou DA, Nouri R, Beheshti S, Sakr S (2015) DREAM: distributed RDF engine with adaptive query planner and minimal communication. *VLDB* 8(6):654–665. <http://www.vldb.org/pvldb/vol8/p654-Hammoud.pdf>
- Harbi R, Abdelaziz I, Kalnis P, Mamoulis N (2015) Evaluating SPARQL queries on massive RDF datasets. *VLDB* 8(12):1848–1851. <http://www.vldb.org/pvldb/vol8/p1848-harbi.pdf>
- Hasan A, Hammoud M, Nouri R, Sakr S (2016) DREAM in action: a distributed and adaptive RDF system on the cloud. In: Proceedings of the 25th international conference on World Wide Web, WWW 2016, Montreal, 11–15 Apr 2016, Companion volume, pp 191–194. <http://doi.acm.org/10.1145/2872518.2901923>
- Jones ND (1996) An introduction to partial evaluation. *ACM Comput Surv (CSUR)* 28(3):480–503
- Neumann T, Weikum G (2008) RDF-3X: a RISC-style engine for RDF. *VLDB* 1(1):647–659
- Peng P, Zou L, Özsu MT, Chen L, Zhao D (2016) Processing SPARQL queries over distributed rdf graphs. *VLDB J Int J Very Large Data Bases* 25(2):243–268
- Potter A, Motik B, Nenov Y, Horrocks I (2016) Distributed RDF query answering with dynamic data exchange. In: International semantic web conference. Springer, pp 480–497
- Sakr S, Al-Naymat G (2010) Relational processing of rdf queries: a survey. *ACM SIGMOD Rec* 38(4): 23–28
- Shi J, Yao Y, Chen R, Chen H, Li F (2016) Fast and concurrent RDF queries with RDMA-based distributed graph exploration. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). USENIX Association, pp 317–332
- Valiant LG (1990) A bridging model for parallel computation. *Commun ACM* 33(8):103–111
- Wang X, Wang J, Zhang X (2016) Efficient distributed regular path queries on RDF graphs using partial evaluation. In: Proceedings of the 25th ACM international conference on information and knowledge management. ACM, pp 1933–1936
- Wu B, Zhou Y, Yuan P, Jin H, Liu L (2014) SemStore: a semantic-preserving distributed RDF triple store. In: CIKM, pp 509–518. <http://doi.acm.org/10.1145/2661829.2661876>
- Wylot M, Cudré-Mauroux P (2016) Diplocloud: efficient and scalable management of RDF data in the cloud. *IEEE Trans Knowl Data Eng* 28(3):659–674. <https://doi.org/10.1109/TKDE.2015.2499202>
- Wylot M, Pont J, Wisniewski M, Cudré-Mauroux P (2011) dipLODocu[RDF]: short and long-tail rdf analytics for massive webs of data. In: Proceedings of the 10th international conference on the semantic web, ISWC'11 – volume Part I. Springer, Berlin/Heidelberg, pp 778–793. <http://dl.acm.org/citation.cfm?id=2063016.2063066>
- Yuan P, Liu P, Wu B, Jin H, Zhang W, Liu L (2013) TripleBit: a fast and compact system for large scale RDF data. *Proc VLDB Endow* 6(7): 517–528
- Zhang X, Chen L, Tong Y, Wang M (2013) EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud. In: 29th IEEE international conference on data engineering, ICDE 2013, Brisbane, 8–12 Apr 2013, pp 565–576. <https://doi.org/10.1109/ICDE.2013.6544856>
- Zou L, Özsu MT, Chen L, Shen X, Huang R, Zhao D (2014) gStore: a graph-based SPARQL query engine. *VLDB J* 23(4):565–590. <https://doi.org/10.1007/s00778-013-0337-7>

Navigational Queries

► Graph Path Navigation

Near-Data Processing

► Active Storage

Network Anomaly Detection

► Big Data in Network Anomaly Detection

Network Big Data Security Issues

Niranjan K. Ray, Biswaranjan Acharya, and Anil Kumar Swain

Kalinga Institute of Industrial Technology (KIIT), Deemed to be University, Bhubaneswar, India

Definitions

- Mobile ad hoc network (MANET): It is a type of wireless networks which is most suitable for environment where establishing other types of networks are neither possible nor advisable. A node in MANETs performs the role of both host as well as router.

- Cloud computing: It is a type of Internet-based computing platform, where different services such as storage, database, integration, security, and management are delivered to computers and devices through the internet. It stores and maintains the data in a cloud data center. Cloud center usually supports more numbers of users, applications, and data.
- Internet of things (IoT): It is the collection of different things (physical gadgets, vehicles, home machines, etc.) inserted with hardware, programming, sensors, and actuators. It can be acknowledged in three standards: web situated (middleware), things arranged (sensors), and semantic-arranged (information).

Introduction

Today a wide range of wireless products are available in the market. They are omnipresent and are influencing the society in many ways. Their applications are ranging from Bluetooth (IEEE 802.15.1), cellular data services (3G/4G), wireless fidelity (IEEE 802.11), Zigbee (IEEE 802.15.4), WiMax (IEEE 802.16), etc. Basically, these technologies are available in different forms such as mobile notebook/tablets and laptops. Most of the wireless networks such as wireless LAN and cellular networks are infrastructure based. They require base station or access point to operate. However, few networks such as MANETs do not require any preestablished infrastructure to operate. Infrastructure-less networks can be deployed anywhere with minimal cost and effort. Other than this there are also many other types of networks present to overcome some of the major challenges faced by the traditional wireless networks. These networks are software-defined networks (SDN), wireless mesh network, and Internet of things (IoT). All these networks are the backbone of many applications such as smart home, smart healthcare, smart city, etc. (Mohanty et al. 2016; Lacink and Ristve 2017; Sundaravadivel et al. 2018). With the increase in the networking applications, there is an increase in data transmission and processing.

This leads to use of big data applications and tools. Networking for big data is still a huge challenging area for researchers. There are many problems that need to be addressed properly such as information retrieval in big data, data storage management for big data, network optimization for big data, network security and privacy issues in big data, and so on. This chapter discusses some security aspects of networking technology used in field in big data applications.

Security Aspects in Networking

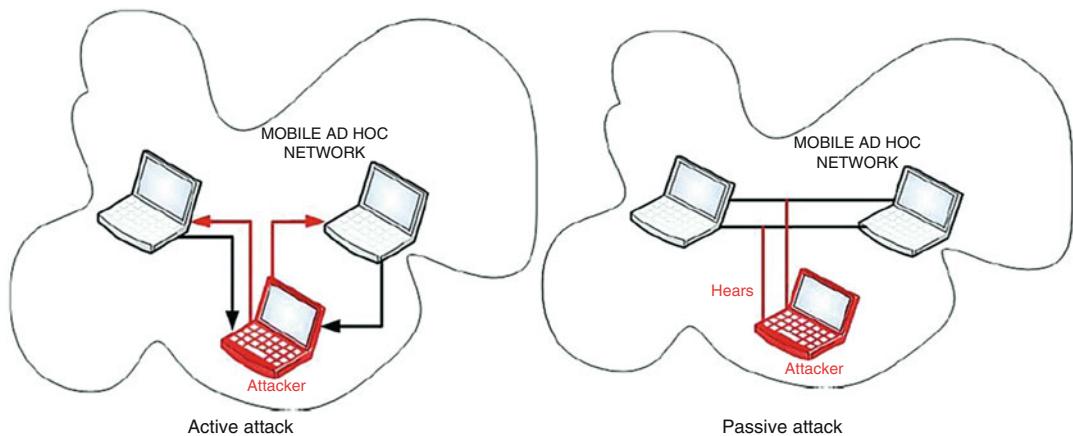
In this section few security mechanisms for different networking applications are discussed briefly.

Security in MANET

Security always remains an important concern in MANET due to its vulnerable characteristics and behavior. Attackers target to attack across all layers of protocol stack. It is observed that different mechanisms are used for this Joshi (2011) and Obi (2004). In Shi-Chang et al. (2010), the security architecture is divided into five layers such as (i) infrastructure layer, (ii) communication security layer, (iii) routing security layer, and (v) application security layer. Specially network layer is severely affected by attackers. Network layer attack in MANETs can be divided into two main categories: (i) active attack and (ii) passive attack. Active attacks are targeted to prevent services from working properly or shut them down completely. Intrusion prevention measures like encryption and authentication can prevent nodes from disrupting traffic. Passive attacks are typically more severe attacks; it is also known as internal attacks. In this attack, a malicious node affects the route discovery process by changing the content of the discovered route. It can also invalidate the route cache of other nodes by advertising incorrect paths. The active and passive attacks are shown in Fig. 1.

Some of the active attacks and their prevention mechanisms are discussed here briefly:

Sleep Deprivation Attack: It is a type of denial-of-service (DoS) attack. In this approach an



Network Big Data Security Issues, Fig. 1 Active and passive attacks in MANETs

attacker tries to interact with a legitimate node to feel that the attacker is a legitimate user, but the main purpose of the attacker is to keep the victim node from entering into the sleep mode. A malicious node forwards a route request (RREQ) packet to all its neighboring node with an address which is not valid in that network.

Black Hole Attack: The intruder responds back RREQ with a packet having large sequence number. It means it has the fresh route to the destination node. By doing this the intruder becomes part of many routes. This way it acts as a relay node in many paths and drops the packet instead of forwarding. Causing serious degradation of throughput.

Gray Hole Attack: It is a type of black hole attack, but it is more difficult to detect than the black hole attack. It operates in two phases: (i) malicious node advertises itself as having a valid route to a destination node, and (ii) the node drops the intercepted packets with a certain probability. The major difference is that in black hole attack, the intruder drops every packet received by the node. However, in gray hole attack, the intruder might drop packet selectively.

Worm Hole Attack: In this approach an attacker tunnels the RREQ directly to the destination node. It prevents any other route from being discovered. The tunnel can be created by means of a high-quality wireless link. After

building a wormhole link, the attacker is able to receive all the packets and forwards them to the other malicious node through the wormhole link.

Sybil Attack: In this approach an attacker creates a new identity, discarding the previously created one. A malicious node tries to gain more resources, information, access, etc. than what a single node deserves in a network.

One solution to overcome sleep deprivation attack (Yi et al. 2009) is neighbor supervision. The neighbor node maintains a priority queue of incoming RREQs and reduces the probability of processing RREQs from a node if a high number of incoming RREQs are received from that node. Few active attacks are discussed in Abdelhaq et al. (2011). On the prevention side, various key and trust management schemes have been developed to prevent external attacks from outsiders, and various secure MANET routing protocols have been proposed to prevent internal attacks originated from within the MANET system (Zhang and Lee 2005).

Intrusion Detection in MANETs

An intrusion detection system (IDS) is a defense system that detects hostile activities in a network. It then tries to prevent many activities that may compromise system security. Intrusion prevention techniques alone may not be sufficient in MANETs. As the system becomes more complex, there are also more weaknesses, which

lead to more security problems. The prevention part may involve issuing alerts as well as taking direct preventive measures such as blocking a suspected connection. Firewalls are the first line of defense; IDSs come into the picture only after an intrusion has occurred and a node has been compromised. IDS can be classified as (i) host based or (ii) network based. In network-based approaches, capture and analyze packets from network traffic in host-based approaches take the help of operating system in its analysis. Few intrusion detection techniques for MANETs are discussed below here.

Stand-Alone Intrusion Detection Systems: In this approach a node independently makes the decision based on the information collected by it to determine the intrusion. It is suitable in the network where nodes are not capable of running an IDS or have an IDS installed in it.

Distributed and Cooperative Intrusion Detection Systems: Node in the network participates in intrusion detection. In this approach, an IDS agent runs at each node. The role of IDS agent is to detect and collect local events and data to identify possible intrusion. The IDS agent also cooperatively participates in global intrusion detection action.

Security in Cloud Computing

Cloud computing is one of the emerging technology in the recent times which has varieties of applications at different fields. In this section security requirements and some security techniques for cloud computing are discussed.

Security Requirements in Cloud Computing

- Authentication: It is a basic requirement to provide security. It identifies the users according to verifying their username and password.
- Authorization: It ensures referential integrity of a task in cloud computing. This process exerts control and privileges over different ongoing process or task.
- Confidentiality: It gives more security to cloud computing and allows only an authorized person to access the data. In cloud computing, this process controls organization's data across the multiple distributed databases. It is employed in public cloud because public

cloud has many numbers of users and it is open for all users. It allows various protocols to be imposed in different layers of cloud applications.

- Integrity: It means the information of a user data cannot be modified by the environment. It assured that information cannot be tampered at multiple distributed databases in different places.
- Non-repudiation: It gives security against false denial of involvement of a communication. Non-repudiation applies the traditional e-commerce security protocols and token provisioning to data transmission within cloud applications such as digital signatures, timestamps, and confirmation receipts.

Security Issues in Cloud Computing

• Software security issues: Software security plays a vital role in computer system at present times. Security measures might be hard because usually software has thousands or millions lines of code. In case of real-time system, fully reliable software is needed to handle the critical situations. A business organization stores their important files and data into digital files in high-value price of information. Here the company revenues in the stake. By software faults, the SaaS providers are to ensure about no data leakage. In spite of being in a more tightly managed environment, software is no more secure simply by virtue of being in a virtualized environment.

- Virtual machine security: Virtualization is a key element of a cloud. Virtualization has a major task – it ensures different instances running in the same physical machine. Without much difficulty virtualization can be cloned and smoothly moved between physical servers. This advantage of VM makes it difficult to achieve and maintain consistent security. At any point of time, it is very hard to maintain the table of VM auditable record of the security. Due to its flexibility, it has wide adoption in the industry. IaaS providers in their business put trust and confidence on quick deployments of VM on top of VMMs. A multi-tenant and virtualized approach gives

profit by cloud providers' perceptive, but it increases the collocation attack surface. Virtualization software contains bugs that allow virtualized code to break loose to some extent. Managing images, monitoring virtual machines, and virtual machine mobility are the issues related on virtualization.

- Internet and services security issues: Cloud infrastructures are composed by the hardware, where data is stored and processed. It is also composed by the path, where it is transmitted. A large number of data packets are transmitted from source to destination by so many numbers of third-party devices and links in a cloud scenario. The transmission medium is the Internet. Some known issues are IP spoofing, port scanning, packet sniffing, malware, and social engineering.
- Network security issues: Network security issue is another key issue in cloud computing applications. Each network is affected by some security threats. Different types of attacks are associated with the networks such as DNS attacks, sniffer attacks, etc. In order to convince the information security and data integrity, Hypertext Transfer Protocol Secure (HTTPS) and Secure Shell (SSH) are the most common acceptance.
- Access security issues: Authentication is a technique which gives security to the data, email, and other accounts of the customer. By combination of username and password, it gives protection. This technique is used in cloud. It is also very much required to deploy security policies to address the issues such as authentication and authorization requirements. For specific access conditions, SaaS service providers' application must be customized and configurable.
- Trust security issues: Trust is one of the security aspects of cloud computing. Cloud providers must trust customers to access their cloud in an honest manner. There are some trust assets that are needed when adding stakeholders in the cloud. Trust is not always established at both customer's and providers' mind. Only trust cannot provide comfort to a customer. Some additional features are needed

to include in order to increase customer's confidence. Trust management is not only a factor in cloud computing, but also it applies in other distributed systems and peer-to-peer (P2P) and sensor networks.

5G Security

The emerging technological concept of the 5G network is promising to fulfill the growing desires of wireless communication. 5G will provide many benefits, such as enhanced speed, lower latency, and better efficiency. However, due to increased numbers of devices and a use of virtualization and cloud, it will equate to many more security threats. At the side of growing speed of transmission, wide variety of customers, reliability, and coverage of the cell network, safety is an issue of key significance that requires careful consideration. The security in 5G basically focuses on the following:

- (i) Prevent threats by using firewalls to protect the network and using access control to minimize user-based risk.
- (ii) Stop and fix advanced malware by behavior-based checks on endpoints.
- (iii) Detect anomalies by using packet capture, big data, and machine learning to identify threats not spotted by basic filters.
- (iv) Incorporate DNS intelligence by monitoring DNS activity and protect against any malicious activities.
- (v) Make threat intelligence paramount by understanding the malicious efforts of hackers.

New security issues beginning in the innovations to be conveyed in 5G were examined to take suitable countermeasures for the future network.

There are numerous advancements proposed in literature that offer arrangements for the 5G systems, e.g., physical layer security, monstrous MIMO, incorporated SDN engineering, etc. (Hidano et al. 2015). There are crucial issues in the 5G where remote transmissions are inalienably helpless against security breaks and among different advancements. The three most encouraging ones are talked about: heterogenous

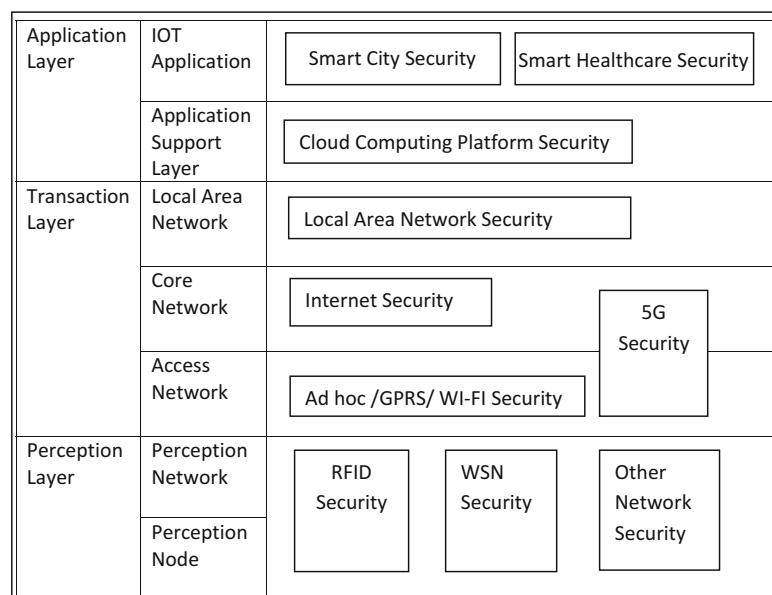
systems, huge numerous information various yield, and millimeter wave (Yang et al. 2015). On the premise of the key standards of every innovation, it recognized the rich open doors and the exceptional difficulties that security architects must handle. Such a recognizable proof is required to conclusively propel the comprehension of future physical layer security. Critical advances will be the illumination of the security prerequisites; the audit of existing security models, specifically the LTE security engineering; and lastly the choice of the 5G safety efforts in tight systems administration with the plan of the general 5G organize design (Schneider and Horn 2015). Physical layer security of applying non-orthogonal various access (NOMA) in huge-scale systems is researched (Qin et al. 2016). A new physical layer paradigm for secure key exchange between the legitimate communication parties in the presence of a passive eavesdropper (Mazin et al. 2017). Another work exhibited agent cases of potential dangers and assaults against the primary parts of the up and coming 5G correspondence frameworks with a specific end goal to explain the future security issues and difficulties in the up and coming 5G period. Especially, it concentrated

on cases of potential dangers and assaults for the accompanying 5G framework parts: the UE, the entrance organizes the versatile administrator's center system and the outer IP systems (Mantas et al. 2015).

Security in IoT

IoT is the collection of different real-world things connected by Internet. It can be acknowledged in three standards: web situated (middleware), things arranged (sensors), and semantic-arranged (information). The association of physical things to the Internet makes it conceivable to get to remote sensor information and to control the physical world from a separation. The functionality of IoT can be expressed in three layers: (i) application layer, (ii) transportation layer, and (iii) perception layer. Application layer security can be achieved by implementing security features in storage servers. Few such work is reported in Sicari et al. (2016), Matharu et al. (2014), Weber (2010), and Nguyen et al. (2015). Transport layer security can be implemented through access network and core network and perception layer security through perception node and network. Functionality and security level at each layer is shown in Fig. 2.

Network Big Data Security Issues, Fig. 2
Security architecture in IoT



Conclusion and Future Direction

Security is considered to be a very big issue in networking. In this chapter we have discussed few security aspects in MANETs, cloud computing, 5G, and IoT application areas. Numerous methods have been proposed in literature to make network more secured. However, as the requirement is changing rapidly and many new applications, protocols, and devices are involved with the network, security will remain the key concern for new technology and applications.

Cross-References

- ▶ [Mobile Big Data: Foundations, State of the Art, and Future Directions](#)
- ▶ [Network-Level Support for Big Data Computing](#)

References

- Abdelhaq M, Serhan S, Alsaqour R, Hassan R (2011) A local intrusion detection routing security over MANET network. In: Electrical engineering and informatics (ICEEI), 2011 international conference on, IEEE, pp 1–6
- Hidano S, Pecovsky M, Kiyomoto S (2015) New security challenges in the 5g network. In: International symposium on intelligence computation and applications, Springer, pp 619–630
- Joshi P (2011) Security issues in routing protocols in MANETs at network layer. Procedia Comput Sci 3:954–960. <https://doi.org/10.1016/j.procs.2010.12.156>. <http://www.sciencedirect.com/science/article/pii/S1877050910005314>, world conference on information technology
- Lacink M, Ristve J (2017) Smart city, safety and security. Procedia Eng 192(Supplement C):522–527
- Mantas G, Komninos N, Rodriguez J, Logota E, Marques H (2015) Security for 5G communications. In: Rodriguez J (ed) Fundamentals of 5G mobile networks. Wiley, pp 207–220. ISBN:9781118867464. <https://www.wiley.com/en-gb/Fundamentals+of+5G+Mobile+Networks-p-9781118867525>
- Matharu GS, Upadhyay P, Chaudhary L (2014) The internet of things: challenges & security issues. In: Emerging technologies (ICET), 2014 international conference on, IEEE, pp 54–59
- Mazin A, Davaslioglu K, Gitlin RD (2017) Secure key management for 5g physical layer security. In: Wireless and microwave technology conference (WAMICON), 2017 IEEE 18th, IEEE, pp 1–5
- Mohanty SP, Choppali U, Koulianios E (2016) Everything you wanted to know about smart cities. IEEE Consumer Electron Mag 5(3):60–70
- Nguyen KT, Laurent M, Oualha N (2015) Survey on secure communication protocols for the internet of things. Ad Hoc Netw 32:17–31
- Obi OO (2004) Security issues in mobile ad-hoc networks: a survey. In: The 17th White House papers graduate research in informatics at Sussex
- Qin Z, Liu Y, Ding Z, Gao Y, Elkashlan M (2016) Physical layer security for 5g nonorthogonal multiple access in large-scale networks. In: Communications (ICC), 2016 IEEE international conference on, IEEE, pp 1–6
- Schneider P, Horn G (2015) Towards 5g security. In: Trustcom/BigDataSE/ISPA, 2015 IEEE, IEEE, vol 1, pp 1165–1170
- Shi-Chang L, Hao-Lan Y, Qing-Sheng Z (2010) Research on MANET security architecture design. In: Signal acquisition and processing, 2010. ICSAP'10. International conference on, IEEE, pp 90–93
- Sicari S, Rizzardi A, Miorandi D, Cappiello C, Coen-Porisini A (2016) A secure and quality-aware prototypical architecture for the internet of things. Inf Syst 58:43–55
- Sundaravadivel P, Koulianios E, Mohanty SP, Ganapathiraju MK (2018) Everything you wanted to know about smart health care: evaluating the different technologies and components of the internet of things for better health. IEEE Consumer Electron Mag 7(1): 18–28. <https://doi.org/10.1109/MCE.2017.2755378>
- Weber RH (2010) Internet of things—new security and privacy challenges. Computer Law Sec Rev 26(1): 23–30
- Yang N, Wang L, Geraci G, Elkashlan M, Yuan J, Di Renzo M (2015) Safeguarding 5g wireless communication networks using physical layer security. IEEE Commun Mag 53(4):20–27
- Yi P, Tong T, Liu N, Wu Y, Ma J (2009) Security in wireless mesh networks: challenges and solutions. In: Information technology: new generations, 2009. ITNG'09. Sixth international conference on, IEEE, pp 423–428
- Zhang Y, Lee W (2005) Security in mobile ad-hoc networks. In: Ad hoc networks, Springer, pp 249–268

Network Data Models

- ▶ [Graph Data Models](#)

Network Databases

- ▶ [Graph Data Models](#)

Network Measurements

- ▶ [Big Data in Computer Network Monitoring](#)

Network Normal Traffic/Behavior Modeling

- ▶ [Big Data in Network Anomaly Detection](#)

Network Outlier Detection

- ▶ [Big Data in Network Anomaly Detection](#)

Network Visualization

- ▶ [Graph Visualization](#)

Network-Level Support for Big Data Computing

Fábio Diniz Rossi¹ and Guilherme da Cunha Rodrigues²

¹Federal Institute of Education Science, and Technology Farroupilha (IFFAR), Alegrete, Brazil

²Federal Institute of Education Science and Technology Sul-Rio Grandense (IFSUL), Charqueadas, Brazil

Introduction

Currently, a huge amount of data has been created from several distributed sources. In this scenario,

a new problem has emerged: how to develop and deploy infrastructures (i.e., storage, network, processing) that are scalable and elastic enough to handle this massive amount of data in a suitable way. The big data concept is related to the capacity of such infrastructures to cope with this enormous amount of data along with quality of service (QoS) metrics that include performance, timeliness, and availability (Assunção et al. 2015).

Scalability is the capacity to enhance the infrastructure by increasing the number of computational resources at the same pace that the quantity of data to be processed grows. It means that the infrastructure must be flexible enough to grow based on the demand for computational resources in order to provide high-quality services (Rodrigues et al. 2016).

To deliver big data demands, the whole infrastructure must be elastic as well. In other words, the infrastructure must grow in quantity of computing resources in order to handle the amount of data that arrives to be processed and reduce such amount of resources when they are not needed. On the one hand, such increase in amount of resources at the infrastructure is necessary to keep quality of service metrics. On the other side, the reduction of resources is necessary to reduce costs when they are no longer necessary.

From the above, big data computing is a complex concept that is dependent on computational resources delivered in a scalable and elastic way. In this context, network resources play a key role in enabling quick and sustainable communications expansion while ensuring that applications are linked to mission-critical data transaction and content environments (Borovick and Villars 2012). This paper presents a definition of network-level support for big data computing along with some important technologies employed to provide support for big data environments and applications.

N

Definitions

A network supporting big data is composed of both links inside data centers (i.e., internal links or internal connections) and links across

the WAN (i.e., external links or external connections). Network-level support for big data computing enables communication among nodes playing a vital role in allowing fast and sustainable exchange of an enormous amount of data (i.e., raw data, analyzed data) as well as ensuring that these nodes are linked to essential systems and applications.

The network infrastructure that could be physical or virtual must be designed to be scalable, elastic, and resilient in order to optimize performance due to the characteristics of big data environments. Thus, the network infrastructure must enable full link utilization by deploying multiple paths allowing the nodes to pick the most efficient route up and in this way offering adequate levels of performance.

Overview

The concept of big data demands that a massive amount of data is processed, generating information in real time. In order to support such processing power regarding response time, large-scale environments must offer some features, such as scalability, availability, and fault tolerance. Such features necessarily pass through a reliable network level to handle all data flows among the processor nodes.

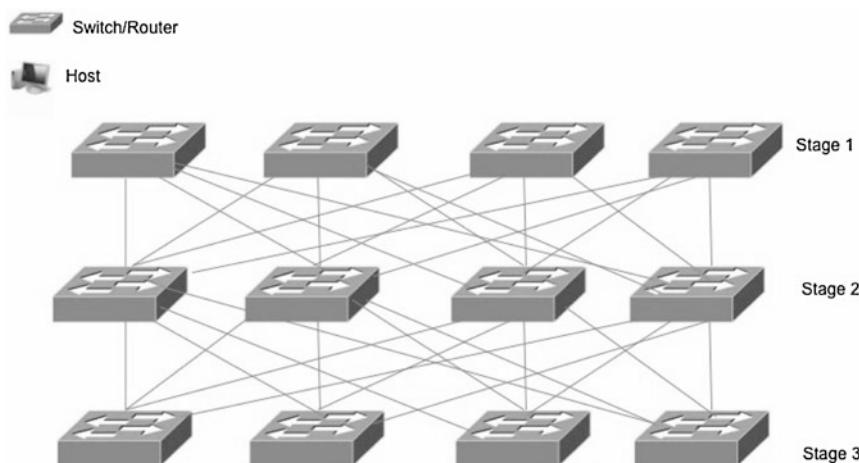
First, this section presents the physical substrate of the network, which consists of the communication devices and channels currently used in large-scale environments that support big data applications. Next, this section presents the software layer that comprises of technologies that make the network flexible enough to fit the application needs. Finally, this section is concluded with a current scenario of convergence between all the network capabilities discussed.

Physical Network Substrate

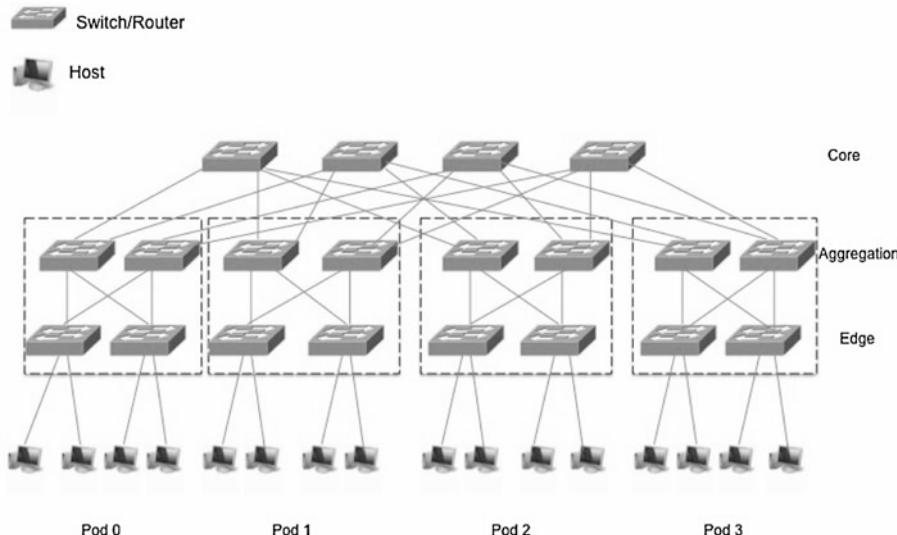
Data centers have become a useful communication infrastructure to support big data. Currently, there are some different ways to deploy a network topology to support big data, but the vast majority are structures based on trees, which includes Clos topology and fat-tree topologies (Al-Fares et al. 2008; Bari et al. 2013; Guo and Yang 2015).

Clos topology is built up from multiple stages of switches as depicted in Fig. 1, which is an example of a Clos topology composed of three stages. This topology is easy to understand, and each switch in a particular stage is connected to all switches in the next stage providing an extensive diversity of paths for data communication.

According to the literature, fat-tree topology is a particular sort of Clos topology that has its structure based on a tree as presented in Fig. 2. Fat-tree topology is built of k-port switches that



Network-Level Support for Big Data Computing, Fig. 1 Clos topology



Network-Level Support for Big Data Computing, Fig. 2 Fat-tree topology ($k = 4$)

contain k pods. Each pod has two levels of $k/2$ switches, namely, aggregation and edge. Each of the $(k/2)$ core switches has one port connected to each of k pods. The i -th port of any core switch is connected to pod i so that consecutive ports in the aggregation level of each pod switch are connected to core switches on $k/2$ strides. Each edge switch is directly connected to $k/2$ end hosts. Each of the remaining $k/2$ ports of an edge switch is connected to $k/2$ ports of an aggregation switch.

Although the physical network substrate is implemented not to be changed, it supports flexibility in many ways. Link aggregation (Seifert 2000) at the network-level consists of a technology that provides means for traffic to be forwarded in parallel through available channels, where each device may dynamically aggregate the corresponding ports into an aggregation port which shows up as a single, logical, high bandwidth port or interface to other processes executing on the device. Currently, it allows link aggregation from two to eight physical channels into a single virtual channel aiming to increase the bandwidth and, as a consequence, increasing performance.

Nowadays, data centers have taken advantage of network devices that provide support to link aggregation. Figure 3 presents the physical con-

nnections of a modern data center network in which the communicating hosts (a) are connected to edge devices (b). Such devices are connected to the network core (c) by multiple channels.

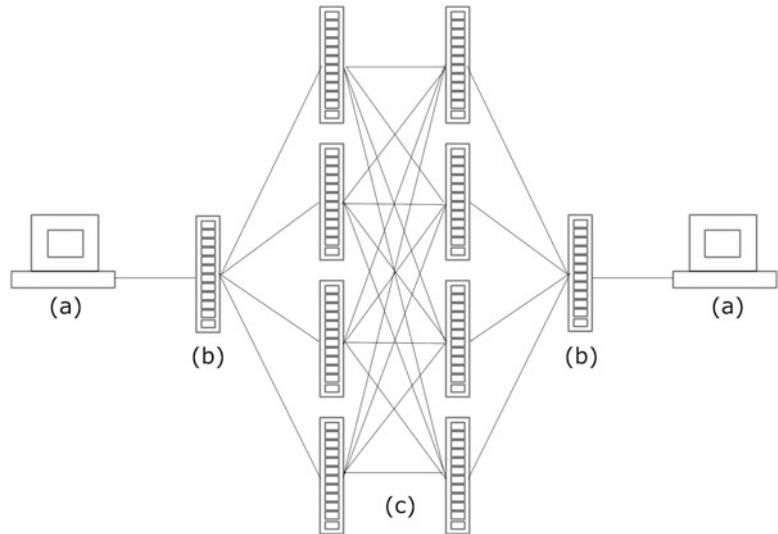
The devices composing the core are interconnected via multiple channels as well. Multiple connections between edge devices and core devices are usually a way to enable routing and high availability. Besides, several protocols have used this technology to support link aggregation. Furthermore, it is important to mention that in case of failure of one of the redundant links, frames may be forwarded to and from the remaining links of the aggregation port transparently to the process executing on the devices.

In this scenario, each network equipment manufacturer has developed its own layer 2 technology (i.e., multi-link trunking) aiming to provide support to link aggregation protocols. In this way, protocols at layer 3 which include virtual concatenation (VCAT) and link aggregation control protocol (LACP) can be developed to enhance both link aggregation and port aggregation (Berggreen and Litjens 2006).

Although this option to use link aggregation is offered to a plethora of data centers, it is not largely utilized because of the enormous traffic on the network, which increases the problem

Network-Level Support for Big Data Computing

Fig. 3 Multi-link network data center substrate



of packet collisions and network loops. Aiming to overcome such limitations, it becomes necessary the use of a technology that manages the aggregation and disaggregation of links with a wide vision of the packet flows. In this context, two innovative technologies have emerged: network function virtualization (NFV) and software-defined networking (SDN).

Logical Network Layer

For a long time, network infrastructures had been limited by physical substrate choices such as equipment and topology. In such environments, the higher level the infrastructure, the more difficult it would be to implement any change. It remained true until the emergence of virtualization. The virtualization layer has provided flexibility over the physical substrates as it allows virtual overlay networks to be maintained, and data streams can be managed independently and in separate paths. Today's network environments use a variety of virtualization models, ranging from traditional virtualization layers such as Xen (Barham et al. 2003) to lightning virtualization layers such as Containers (Xavier et al. 2013) and Unikernels (Xavier et al. 2016), each one being chosen due to its capabilities, such as resource isolation, performance, and security, among others.

Based on this new idea, specialized appliances were replaced by virtual machines, in order to speed up the deployment of network services. As a consequence, dedicated appliances were replaced by virtual machines, with the goal of speeding up the deployment of network services to meet the growth of data centers. This new concept is known as network function virtualization (NFV) (Mijumbi et al. 2016), and it allows network services to be started and stopped according to demand. Therefore, NFV services provide the ability to tune and modify network resources in real time, delivering customized customer experiences. It makes the infrastructure flexible enough to evolve at the same pace as the applications and their related services.

One technology that has been developed side by side, but independently from NFV, is software-defined networking (SDN) (Lemeshko and Vavenko 2017). It is a technology that offers dynamicity, manageability, and adaptive computer networking to big data environments. It is currently one of the most promising technologies for handling the high bandwidth requirements and the dynamic nature of big data applications, making networks more flexible and efficient.

In SDN, the network control is programmable directly by administrators, enabling support to key functions that include configuration, management, and protection. In this way, SDN

optimizes the use of network resources in a quick and efficient manner. Additionally, such a programmable network provides support to administrators to dynamically adjust the flow of data traffic through the network to meet the demands of change since all network intelligence is logically centralized in software-based controllers that have a global view of the network.

It is important to mention as well that, in SDN, the data plane is separated from the control plane, thus enabling a new operational method that is different from the traditional computer network. Figure 4 depicts such separation between data plane and control plane showing their interaction through an orchestrator. Traditional computer networks have proposed that each device has the data plane and the control plane. On the other side, SDN has proposed to separate both the control plane and the data plane. The control plane is centralized in the SDN controller, and the data plane is kept on the network device.

The control plane (Zhao et al. 2017) is responsible for understanding how the packets have been routed, in other words, what is the best route to forwarding the data flow and how to better assemble the routing table, whereas the data plane is responsible for routing packets based on simple rules, which is associated with each entry in the routing table of the network device.

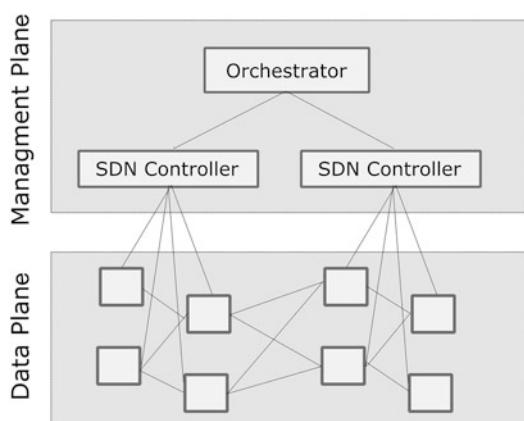
One important characteristic that makes SDN a technology viable to the needs of big data applications is that all packet forwarding requests are made from the hosts to the SDN controller. Thus, the controller knows in advance which path these packets will use to traffic, and so it can expand the channels of that path at runtime before sending the devices the route to be established. Such characteristic is significant when dealing with a huge amount of data usually forwarded in big data environments.

Finally, SDN controllers can respond to requests from network devices and reconfigure them quickly if compared to traditional infrastructures, with the advantage of adapting to network growth transparently. Thus, it provides an efficient method to improve performance as well.

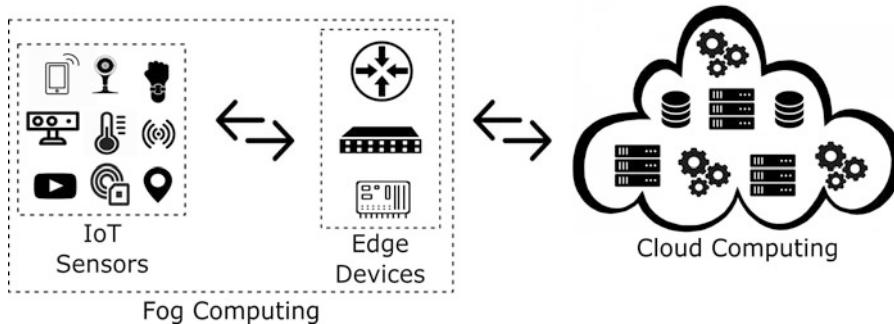
Network Convergence

In big data environments, cloud providers offer data analysis and integration services in the form of an Analytics-as-a-Service (AaaS) (Lomotey and Deters 2014). The exponential increase in a number of sensor-generating data to be processed by such services boosted the advent of the Internet-of-Things (IoT) (Dlamini and Johnston 2016). To meet this growing demand in the amount of data to be transmitted between the customer and the cloud, it would be necessary to increase the capacity of the communication channels. However, the increase in this type of infrastructure is expensive. For a time, techniques similar to offloading (Gao et al. 2017) tried to be a palliative to solve the limitations of latency in the data transfer.

Currently, edge computing (Klas 2017) is a low-cost change in infrastructure that addresses such limitations because it brings cloud services closer to the customer and offer embedded devices with relative processing power. From the above, there is a new emerging trend toward the integration of IoT, cloud computing, and edge devices that is known as fog computing (Yousefpour et al. 2017). Therefore, such new paradigm is defined as an architecture that extends computational capacity and cloud storage to network



Network-Level Support for Big Data Computing, Fig. 4 SDN architecture plane



Network-Level Support for Big Data Computing, Fig. 5 The composition of fog computing and its relation with cloud computing

access layers, allowing data to be analyzed and transformed into information or actions more quickly.

Although some authors treat fog and edge computing as synonyms, there is a difference between these two approaches as to where intelligence is located and the power of processing. Figure 5 illustrates such a difference within a large-scale computing environment, where edge computing brings processing power and communication capabilities to programmable devices located at the edge of the network, such as devices that once played only gateways. It draws closer to the consumer a significant processing power and network, providing quality of service and reducing point of failure. Fog computing extends cloud power by encompassing IoT and edge computing devices in an environment of distributed and scalable services, supported by processor nodes, networking, and storage.

Therefore, fog computing has emerged based on several different technologies such as sensor networks, ubiquitous computing, pervasive computing, grids, and cloud computing. It is also thankful for the spread of customizable low-cost hardware with a variety of modules to allow exchange of information of these platforms with other devices. While this promotes a high adaptability to many environments and contexts, it is weakened by the lack of standardization in communication. In particular, networks that serve as a communication channel between these devices have not been developed specifically for

this type of scenario. Therefore, fog has used existing technologies, and these do not take into account the characteristics of the sensors, for example, concerning about energy waste.

Examples of Network-Level Application for Big Data Processing

Among the applications that provide environments at the network infrastructure level for big data processing, Containers are the ones that cause the most reliability. Containers (Xavier et al. 2016) consist of lightweight virtual machines, which, unlike traditional virtual machines, perform similarly to applications running directly on the operating system. It is because instead of virtualization being managed by an additional layer called hypervisor, Containers are managed directly by the kernel.

This type of virtualization is standard in large-scale environments such as Mesos (Delvalle et al. 2016), Docker (Dirk 2014), and YARN (Vavilapalli et al. 2013). Scalable applications such as Hadoop (Mallika and Selvamuthukumaran 2017) and MPI (Subramoni et al. 2017) can be used for the Containers features, such as elasticity and resource isolation. In this way, big data applications can grow over need-based resources, isolated from one another through virtual network channels. From the service provider's point of view, Containers provide a fast deployment option of the infrastructure tools needed to process big data applications.

Security is another critical topic (Barona and Anita 2017). Many data centers that support private clouds at peak times do not have sufficient resources to meet the demands to be processed while maintaining quality of service for big data applications. In this case, service providers allocate a slice of resources in public clouds. This mechanism, called hybrid clouds, and some security issues must be taken into account, based on the importance of the data that travels out of the private environment of the service provider. In order to maintain control over these issues, some techniques already used at the network level are applied, such as the virtual private network (VPN) and encryption (Mary and Amalarethinam 2017).

Future Directions for Research

Based on the themes presented, some significant research directions may be pointed out.

- The network level consists of consolidated and old technologies. Research on the physical substrate will require a lot of effort with a minimal gain. Searches should be directed to software layers that can modify flows without changing the physical substrate. NFV and SDN can contribute to the attendance of several metrics, such as performance, security, and availability, among others.
- New approaches for virtualization are emerging, but they are still immature to be used in production environments, but that in a short time may replace Containers as a de facto standard in support of big data applications, such as Unikernels.
- Allocating network resources intelligently is a crucial point for improving the performance of big data applications. Some big data programming models such as Hadoop (Mallika and Selvamuthukumaran 2017) exchange data between processors that support map and reduce processes. Therefore, it is interesting that such processes are allocated to nearby nodes in the network. Also, increasing the bandwidth be-

tween such nodes dynamically also becomes an interesting capacity to the network.

- Undeniably, the most massive volume of data currently generated comes from IoT devices. The traffic and analysis of this data depend on open environments that do not limit such growth. Therefore, a concept that is not new but is more current than ever consists of network neutrality (Garrett et al. 2017). Technically, this concept means that any data that travels over the network must be treated in the same way and be transferred at the same speed. With the exponential growth of mobile devices, network neutrality is becoming increasingly difficult. But much more than a technical concept, there is a political and financial motivation in the sense that certain service providers could privilege data from some types of devices or equipment brands to the detriment of others.

References

- Al-Fares M, Loukissas A, Vahdat A (2008) A scalable commodity data center network architecture. ACM SIGCOMM – Comput Commun Rev 38-4(October 2008):63–74
- Assunção MD, Calheiros RN, Bianchi S, Netto MAS, Buyya R (2015) Big data computing and clouds: trends and future directions. J Parallel Distrib Comput 79–80:3–15
- Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. SIGOPS Oper Syst Rev 37(5):164–177
- Bari F, Boutaba R, Esteves R (2013) Data center network virtualization: a survey. IEEE Commun Surv Tutorials 15-2:909–928
- Barona R, Anita EAM (2017) A survey on data breach challenges in cloud computing security: issues and threats, 2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT), Kollam, pp 1–8
- Berggreen F, Litjens R (2006) Performance analysis of access selection and transmit diversity in multi-access networks. Proceedings of the 12th annual international conference on mobile computing and networking, Los Angeles, 23–29 Sept 2006, vol 1, pp 251–261
- Borovick L, Villars RL (2012) The critical role of the network in big data applications, White paper. IDC/CISCO, Framingham
- da Cunha Rodrigues G, Calheiros RN, Guimarães VT, dos Santos GL, de Carvalho MB, Granville LZ, Tarouco

- L, Buyya R (2016) Monitoring of cloud computing environments: concepts, solutions, trends and future directions. Proceedings of the 31st annual ACM symposium on applied computing, Pisa, 4–8 Apr 2016, vol 1–2, pp 378–383
- DelValle R, Rattihalli G, Beltre A, Govin-Daraju M, Lewis M (2016) Exploring the design space for optimizations with Apache Aurora and Mesos, IEEE international conference on Cloud Computing (CLOUD) applications track, 2016
- Dirk M (2014) Docker: lightweight Linux containers for consistent development and deployment. *Linux J* 2014(239):2
- Dlamini NN, Johnston K (2016) The use, benefits and challenges of using the Internet of Things (IoT) in retail businesses: a literature review, 2016 International Conference on Advances in Computing and Communication Engineering (ICACCE), Durban, pp 430–436
- Gao G, Xiao M, Wu J, Han K, Huang L, Zhao Z (2017) Opportunistic mobile data offloading with deadline constraints. *IEEE Trans Parallel Distrib Syst* PP(99):1–1
- Garrett T, Dustdar S, Bona LCE, Duarte EP (2017) Ensuring network neutrality for future distributed systems, 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, pp 1780–1786
- Guo Z, Yang Y (2015) On nonblocking multirate multicast fat-tree data center networks with server redundancy. *IEEE Trans Comput* 64:1058–1073
- Klas G (2017) Edge computing and the role of cellular networks. *Computer* 50(10):40–49
- Lemeshko AV, Vavenko TV (2017) Development and research of the flow model of adaptive routing in the software-defined networks with load balancing. *Proc Tomsk State Univ Control Syst Radioelectron* 29(3):100–108
- Lomotey RK, Deters R (2014) Analytics-as-a-Service (AaaS) tool for unstructured data mining, 2014 IEEE international conference on cloud engineering, Boston, pp 319–324
- Mallika C, Selvamuthukumaran S (2017) Hadoop framework: analyzes workload prediction of data from cloud computing, 2017 international conference on IoT and application (ICIOT). Nagapattinam, India, pp 1–6
- Mary BF, Amalarethinam DIG (2017) Data security enhancement in public cloud storage using data obfuscation and steganography, 2017 World Congress on Computing and Communication Technologies (WCCCT), Tiruchirappalli, Tamil Nadu, pp 181–184
- Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Boutaba R (2016) Network function virtualization: state-of-the-art and research challenges. *IEEE Commun Surv Tutorials* 18(1): 236–262
- Seifert R (2000) The switch book: the complete guide to LAN switching technology, 1st edn. Wiley, New York
- Subramoni H, Lu X, Panda DK (2017) A scalable network-based performance analysis tool for MPI on large-scale HPC systems, 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, pp 354–358
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O’Malley O, Radia S, Reed B, Baldeschwieler E (2013) Apache Hadoop YARN: yet another resource negotiator, Proceedings of the third ACM Symposium on Cloud Computing (SOCC)
- Xavier MG, Neves MV, Rossi FD, Ferreto TC, Lange T, De Rose CAF (2013) Performance evaluation of container-based virtualization for high performance computing environments, 2013 21st Euromicro international conference on parallel, distributed, and network-based processing, Belfast, pp 233–240
- Xavier B, Ferreto T, Jersak L (2016) Time provisioning evaluation of KVM, Docker and Unikernels in a cloud platform, 2016 16th IEEE/ACM international symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, pp 277–280
- Yousefpour A, Ishigaki G, Jue JP (2017) Fog computing: towards minimizing delay in the internet of things, 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, pp 17–24
- Zhao Y, Zhang P, Wang Y, Jin Y (2017) SDN-enabled rule verification on data plane. *IEEE Commun Lett* PP(99):1–1

Next-Generation Sequencing

- [Big Data Technologies for DNA Sequencing](#)

Non-monotonic Snapshot Isolation in Geo-replicated Systems

- [Achieving Low Latency Transactions for Geo-replicated Storage with Blotter](#)

NoSQL Benchmarks

- [CRUD Benchmarks](#)

NoSQL Database Systems

Sherif Sakr

Institute of Computer Science, University of Tartu, Tartu, Estonia

Synonyms

Document store; Extensible record store;
Key-value store

Definitions

NoSQL (Not Only SQL) is a new generation of high-performance database systems that have been designed to deal with the increasing scaling requirement of modern Web-scale applications. In particular, the new NoSQL systems had a number of design features in common:

- The ability to horizontally scale out through-put over many servers.
- A simple call level interface or protocol.
- Supporting weaker consistency models in contrast to ACID guaranteed properties for transactions in most traditional RDBMS. These models are usually referred to as *BASE* models (*Basically Available, Soft state, Eventually consistent*) (Pritchett 2008).
- Efficient use of distributed indexes and RAM for data storage.
- The ability to dynamically define new attributes or data schema.

These design features are made in order to achieve the following system goals (Sakr 2014; Zhao et al. 2014):

- *Availability*: They must always be accessible even on the situations of having a network failure or a whole datacenter that went of-flne.
- *Scalability*: They must be able to support very large databases with very high request rates at very low latency.

• *Elasticity*: They must be able to satisfy changing application requirements in both directions (scaling up or scaling down). Moreover, the system must be able to gracefully respond to these changing requirements and quickly recover its steady state (Sakr et al. 2011b; Sakr and Liu 2013).

• *Load balancing*: They must be able to automatically move load between servers so that most of the hardware resources are effectively utilized and to avoid any resource overloading situations.

• *Fault tolerance*: They must be able to deal with the situation that the rarest hardware problems go from being freak events to eventualities. While hardware failure is still a serious concern, this concern needs to be addressed at the architectural level of the database, rather than requiring developers, administrators, and operations staff to build their own redundant solutions.

• *Ability to run in a heterogeneous environment*: On scaling out environment, there is a strong trend toward increasing the number of nodes that participate in query execution. It is nearly impossible to get homogeneous performance across hundreds or thousands of compute nodes. Part failures that do not cause complete node failure but result in degraded hardware performance become more common at scale. Hence, the system should be designed to run in a heterogeneous environment and must take appropriate measures to prevent performance degradation that are due to parallel processing on distributed nodes.

N

Overview

For decades, relational database management systems (e.g., MySQL, PostgreSQL, SQL Server, Oracle) have been considered as the *one-size-fits-all* solution for data storage and retrieval. They have matured after extensive research and development efforts and very successfully created a large market and solutions in different business domains. However,

ever-increasing need for scalability and new application requirements have created new challenges for traditional RDBMS. Therefore, there has been some dissatisfaction with this *one-size-fits-all* approach in some Web-scale applications (Stonebraker 2008). The *CAP* theorem (Brewer 2000) has shown that a distributed database system can only choose at most two out of three properties: *Consistency*, *Availability*, and *tolerance to Partitions*. Therefore, most of these systems decide to compromise the strict consistency requirement. In particular, they apply a relaxed consistency policy called *eventual consistency* (Vogels 2008) which guarantees that if no new updates are made to a replicated object, eventually all accesses will return the last updated value (Bermbach et al. 2013). If no failures occur, the maximum size of the *inconsistency window* can be determined based on factors such as communication delays, the load on the system, and the number of replicas involved in the replication scheme.

Google has introduced one of the early systems leading the wave of NoSQL systems by presenting *Bigtable* as a distributed storage system for managing structured data that is designed to scale to a very large size (petabytes of data) across thousands of commodity servers (Chang et al. 2008a). Bigtable has been used by many Google products and projects including Google search engine, Google Finance, Orkut, Google Docs, and Google Earth. Bigtable does not support a full relational data model. However, it provides clients with a simple data model that supports dynamic control over data layout and format. In particular, a Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp. Each value in the map is an uninterpreted array of bytes. Thus, clients usually need to serialize various forms of structured and semi-structured data into these strings. A concrete example that reflects some of the main design decisions of Bigtable is the scenario of storing a copy of a large collection of web pages into a single table. At the physical level, Bigtable uses the distributed Google File System

(GFS) (Ghemawat et al. 2003) to store log and data files. The Dynamo system (DeCandia et al. 2007) is another leading system that has been introduced by Amazon as a highly available and scalable distributed key-/value-based datastore. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the trade-offs between availability, consistency, cost-effectiveness, and performance. Dynamo provided a simple primary-key-only interface and relied on a query model that uses simple read and write operations to a data item that is uniquely identified by a key. State is stored as binary objects (blobs) identified by unique keys. No operations span multiple data items. Dynamo's partitioning scheme relies on a variant of consistent hashing mechanism (Karger et al. 1997) to distribute the load across multiple storage hosts.

Following the main concepts of *Google Bigtable* and *Amazon Dynamo*, several systems have been implemented (<http://NoSQL-database.org/>). In general, these systems can be broadly classified into the following categories (Cattell 2011; Sakr et al. 2011a):

- *Key-value stores*: These systems use the simplest data model which is a collection of objects where each object has a unique key and a set of attribute/value pairs.
- *Document stores*: These systems have the data models that consist of objects with a variable number of attributes with a possibility of having nested objects.
- *Extensible record stores*: They provide variable-width tables (column families) that can be partitioned vertically and horizontally across multiple nodes.

Examples of Systems

Key-Value Stores

Key-value stores use a simple data model, in which data are considered as a set key-value pairs, in which keys are unique IDs for each data and also work as indexes during accessing the data. Values are attributes or objects which contain the actual information of data. There-

fore, these systems are called key-value stores. The data in key-value stores can be accessed using simple interfaces such as insert, delete, and search by key. Normally, secondary keys and indexes are not supported. In addition, these systems also provide persistence mechanism as well as replication, locking, sorting, and other features. For example, **Redis** (Zawodny 2009) is an open-source key-value store system written in C. It supports a fairly rich data model for stored values. Values can be lists, sets, and structures in addition to basic types (Integer, String, Double, and Boolean). Apart from ordinary operations such as reading and writing, Redis also provides a few atomic modifier such as increment of a numeric value by one, adding an element to a list, etc. Redis mainly stores data in memory, which ensures high performance. To provide persistence, data snapshots and modification operations are written out to disk for failure tolerance. Redis can scale out by distributing data (normally achieved at client side) among multiple Redis servers and providing asynchronous data replication through master-slaves.

Memcached (Fitzpatrick 2004) is the first generation of key-value stores initially working as cache for web servers then being developed as a memory-based key-value store system. Memcached has been enhanced to support features such as high availability, dynamic growth, and backup. The original design of Memcached does not support persistence and replication. However, its follow-up variation **Membrain** and **Membase** does include these features which make them more like storage systems. **DynamoDB** (Sivasubramanian 2012) is a NoSQL store service provided by Amazon. Dynamo supports a much more flexible data model especially for key-value stores. Data in Dynamo are stored as tables, each of which has a unique primary ID for accessing. Each table can have a set for attributes which are schema free and scalar types and sets are supported. Data in Dynamo can be manipulated by searching, inserting, and deletion based on the primary keys. In addition, conditional operation, atomic modification, and search by non-key attributes are also supported (yet inefficient), which makes it also closer to that of a docu-

ment store. Dynamo provides a fast and scalable architecture where sharding and replication are automatically performed. In addition, Dynamo provides support for both eventually consistency and strong consistency for reads while strong consistency degrades the performance. Project **Voldemort** (<http://www.project-voldemort.com/>) has been contributed by LinkedIn. It provides multi-version concurrency control (MVCC) for updates.

Document Stores

Document stores provide a more complex data structure and richer capabilities than key-value systems. In document stores, the unit of data is called a document which is actually an object that can contain an arbitrary set of fields, values, and even nested objects and arrays. Document stores normally do not have predefined schemas for data and support search and indexing by document fields and attributes. Unlike key-value stores, they generally support secondary indexes, nested objects, and lists. Additionally, some of them can even support queries with constraints, aggregations, sorting, and evaluations. For example, **MongoDB** (<https://www.mongodb.com/>) is an open-source project developed in C++ and supported by the company 10gen. MongoDB provides its data model based on JSON documents and maintained as BSON (a compact and binary representation of JSON). Each document in MongoDB has a unique identifier which can be automatically generated by the server or manually created by users. A document contains an arbitrary set of fields which can be either arrays or embedded documents. MongoDB is schema free and even documents in the same collection can have completely different fields. Documents in MongoDB are manipulated based on the JSON representation using search, insertion, deletion, and modification operations. Users can find or query documents by writing them as expressions of constraints of fields. In addition, complex operations such as sorting, iteration, and projecting are supported. Moreover, users can perform MapReduce-like program and aggregation paradigms on documents, which makes it possible to execute more complicated analytic

queries and programs. Documents can be completely replaced, and any parts of their fields can also be manipulated and replaced. Indexes of one or more fields in a collection are supported to speed up the searching queries. In addition, MongoDB scales up by distributing documents of a collection among nodes based on a sharding key. Replication between master and slaves with different consistency models depends on whether reading from secondary nodes is allowed and how many nodes are required to reach a confirmation.

CouchDB (<http://couchdb.apache.org/>) is an Apache open-source project written in Erlang. It is a distributed document-based store that manipulates JSON documents. CouchDB is schema free; documents are organized as collections. Each document contains a unique identifier and a set of fields which can be scalar fields, arrays, and embedded documents. Queries on CouchDB documents are called views which are MapReduce-based JavaScript functions specifying the matching constraints and aggregation logics. These functions are structured into so-called designed documents for execution. For these views, B-Tree-based indexes are supported and updated during modifications. CouchDB also supports optimistic locks based on MVCC (multi-version concurrency control) Bernstein and Goodman (1981) which enables CouchDB to be lock-free during reading operations. In addition, every modification is immediately written down to the disk and old versions of data are also saved. CouchDB scales by asynchronous replication, in which both master-slave and master-master replication are supported. Each client is guaranteed to see a consistent state of the database; however, different clients may see different states (as strengthened eventually consistency). Amazon provides Simple Storage Service (S3) (<https://aws.amazon.com/s3/>) as an online public storage web service that provides infinite store for objects of variable sizes. In S3, an object is simply a byte container which is identified by a URI. Clients can read and update S3 objects remotely using a simple web service (SOAP- or REST-based) interface.

Extensible Record Stores

Extensible record stores (also called column stores) are initially motivated by Google's Bigtable project (Chang et al. 2008b). In the system, data are considered as tables with rows and column families in which both rows and columns can be split over multiple nodes. Due to this flexible and loosely coupled data model, these systems support both horizontal and vertical partitioning for the scalability purposes. In addition, correlated fields/columns (named as column families) are located on the same partition to facilitate query performance. Normally column families are predefined before creating a data table. However, this is not a big limitation as new columns and fields can always be dynamically added to the existing tables. For example, HBase (George 2011) is an Apache open-source project and is developed in Java based on the principles of Google's Bigtable. HBase is built on the Apache Hadoop Framework and Apache ZooKeeper (Hunt et al. 2010) to provide a column-store database. As HBase is inherited from Bigtable, they share a lot of features in both data model and architecture. However, HBase is built on HDFS (Hadoop Distributed File System) instead of GFS, and it uses ZooKeeper for cluster coordination compared with using Chubby in Bigtable. HBase puts updates in the memory and periodically writes them to disk. Row operations are atomic with the support of row-level transactions. Partitions and distributions are transparent to users, and there is no client-side hashing like some of the other NoSQL systems. HBase provides multiple master nodes to tackle the problem of single-point failure of the master node. Compared with Bigtable, HBase does not have location groups but only that of column families. In addition, HBase does not support secondary indexing; therefore, queries can only be performed based on primary keys or by fully scanning the table. Nevertheless, additional indexes can be manually created using extra tables.

Hypertable (<http://hypertable.org/>) project is designed to achieve a high performance, scalable, distributed storage and processing system for

structured and unstructured data. It is designed to manage the storage and processing of information on a large cluster of commodity servers, providing resilience to machine and component failures. Like HBase, Hypertable also runs over HDFS to leverage the automatic data replication and fault tolerance that it provides. In Hypertable, data is represented in the system as a multidimensional table of information. The Hypertable systems provide a low-level API and Hypertable Query Language (HQL) that allows to create, modify, and query the underlying tables. The data in a table can be transformed and organized at high speed by performing computations in parallel, pushing them to where the data is physically stored.

Cassandra (Lakshman and Malik 2010) is an open-source NoSQL database initially developed by Facebook in Java. It combines the ideas of both Bigtable and Dynamo and it is now open sourced under the Apache license.

Cassandra shares the majority of the features as other extensible record stores (column stores) in both data model and functionality. It has column groups and updates are cached in the memory first then flushed to disk. However, there still some differences:

- Cassandra have columns which are the minimum unit for storage and super columns which contains a set of columns to provide additional nestedness.
- Cassandra is fully decentralized of which every node in the cluster is considered equal and performs identical functions. In Cassandra, a leader is selected based on the gossip protocol, failures are detected by using the phi accrual algorithm, and scalability is achieved by consistent hashing. All the process that have been mentioned before, leader selection, failure detection, and recovery, are performed automatically.
- Cassandra only supports the eventually consistency model. It provides quorum reads to ensure clients get the latest data from majority of the replicas. Writes in Cassandra are atomic within a column family, and some

extent of versioning and conflict resolution are supported.

References

- Bermbach D, Zhao L, Saks S (2013) Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services. In: TPCTC. Springer, pp 32–47
- Bernstein PA, Goodman N (1981) Concurrency control in distributed database systems. ACM Comput Surv (CSUR) 13(2):185–221
- Brewer EA (2000) Towards robust distributed systems (abstract). In: PODC, p 7
- Cattell R (2011) Scalable SQL and NoSQL data stores. SIGMOD Rec 39(4):12–27. <https://doi.org/10.1145/1978915.1978919>
- Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008a) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):4:1–4:26. <https://doi.org/10.1145/1365815.1365816>
- Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008b) Bigtable: a distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 26(2):4
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. SIGOPS Oper Syst Rev 41(6): 205–220. <https://doi.org/10.1145/1323293.1294281>
- Fitzpatrick B (2004) Distributed caching with memcached. Linux J 2004(124):5
- George L (2011) HBase: the definitive guide. O'Reilly Media, Inc., Beijing
- Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. SIGOPS Oper Syst Rev 37(5):29–43. <https://doi.org/10.1145/1165389.945450>
- Hunt P, Konar M, Junqueira FP, Reed B (2010) Zookeeper: wait-free coordination for internet-scale systems. In: USENIX annual technical conference, vol 8, p 9
- Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In: Proceedings of the 29th annual ACM symposium on theory of computing (STOC '97). ACM, El Paso, pp 654–663. <https://doi.org/10.1145/258533.258660>
- Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review 44(2):35–40
- Pritchett D (2008) BASE: an ACID alternative. Queue 6(3):48–55. <https://doi.org/10.1145/1394127.1394128>
- Saks S (2014) Cloud-hosted databases: technologies, challenges and opportunities. Cluster Computing 17(2):487–502

- Sakr S, Liu A (2013) Is your cloud-hosted database truly elastic? In: Proceedings of the ninth IEEE 2013 world congress on services (SERVICES). IEEE, pp 444–447
- Sakr S, Liu A, Batista DM, Alomari M (2011a) A survey of large scale data management approaches in cloud environments. *IEEE Commun Surv Tutorials* 13(3): 311–336. <https://doi.org/10.1109/SURV.2011.032211.00087>
- Sakr S, Zhao L, Wada H, Liu A (2011b) Clouddb autoadmin: towards a truly elastic cloud-based data store. In: Proceedings of the 2011 IEEE international conference on web services (ICWS). IEEE, pp 732–733
- Sivasubramanian S (2012) Amazon dynamodb: a seamlessly scalable non-relational database service. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. ACM, pp 729–730
- Stonebraker M (2008) One size fits all: an idea whose time has come and gone. *Commun ACM* 51(12):76
- Vogels W (2008) Eventually Consistent. *Queue* 6:14–19. <https://doi.org/10.1145/1466443.1466448>
- Zawodny J (2009) Redis: lightweight key/value store that goes the extra mile. *Linux Mag* 79
- Zhao L, Sakr S, Liu A, Bouguettaya A (2014) Cloud data management. Springer, Cham

O

Object Matching

- ▶ Large-Scale Entity Resolution

Object-Centric Process Mining

- ▶ Artifact-Centric Process Mining

OLAP Benchmarks

- ▶ Analytics Benchmarks

OLTP Benchmarks

- ▶ Analytics Benchmarks

OLTP on Modern Hardware

- ▶ Hardware-Assisted Transaction Processing

One-Pass Algorithms

- ▶ Types of Stream Processing Algorithms

Online Algorithms

- ▶ Types of Stream Processing Algorithms

Online Conformance Checking

- ▶ Streaming Process Discovery and Conformance Checking

Online Machine Learning Algorithms over Data Streams

András A. Benczúr¹, Levente Kocsis¹, and Róbert Pálovics²

¹Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA
SZTAKI), Budapest, Hungary

²Department of Computer Science, Stanford
University, Stanford, CA, USA

Synonyms

Incremental learning; Online machine learning
on streams; Stream learning

Definitions

- Data stream algorithms process a continuous stream of data with only a limited possibility to store past records.
- Online machine learning covers methods that update their models after observing a new event and can immediately serve predictions based on the updated model.

Overview

In this chapter, we overview the main online learning methods for classification and regression. We highlight the most important ideas, including linear models, gradient descent, and tree based methods. This chapter is based on the concepts introduced in the “Overview of Online Machine Learning in Big Data Streams” chapter of this Encyclopedia. Additional topics are covered in the chapters “Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning,” and “Recommender Systems over Data Streams”.

Introduction

Classification and regression are the most widely used machine learning tools, with the goal to predict the unknown label of a data instance based on its attributes or variables. The label is discrete for classification and continuous for regression. Classification and regression in batch settings are well established, with several textbooks available in data mining (Pang-Ning et al. 2006) and machine learning (Friedman et al. 2001).

Classification by online machine learning is summarized in one of the earliest overviews of the field (Widmer and Kubat 1996). The principal task is to learn a concept incrementally by processing labeled training examples one at a time. After each data instance, we can update the model, after which the instance is discarded. The requirements for **machine learning over fast data streams** are summarized in the “► [Overview of Online Machine Learning in Big](#)

[Data Streams](#)” chapter of this Handbook: The algorithms should (1) work from data streams with only a limited possibility to store past data, (2) adapt their models to potential concept drift in the data, and (3) operate in a multi-server distributed computational environment.

For classification and regression over streaming data, the first two of the requirements play a key role. By the first requirement, data is transient, since it does not fit in main memory; hence algorithms that iterate over the entire data multiple times are ruled out. By the second requirement, we cannot assume that the examples are independent, identically distributed, and generated from a stationary distribution; hence different predictions by different models have to be applied and evaluated at different times. Special modeling tools are required to meet the two challenges, and known evaluation and comparison methods are not convenient yet (Gama et al. 2009).

An important, and perhaps the oldest, application of online learning is single-trial EEG classification, in which the system learns from the online feedback of the experimental subjects (Obermaier et al. 2001). These early experiments differ from the approach in this section in that the performance was only measured at the end of the experiments, and no systematic analysis of the classifier performance in time was conducted.

This section is organized as follows: First, we describe the difficulties and possibilities of evaluating online learning methods in section “[Evaluation](#).“ We cover the most important online classification and regression methods in the remaining sections. We discuss the main linear models in section “[Linear Models](#),“ tree-based methods in section “[Decision and Regression Trees](#),“ classifier ensembles in section “[Ensemble Methods](#),“ Bayes models in section “[Bayes Models](#),“ and finally, neural networks in section “[Neural Networks](#).“ Other methods such as **nearest neighbor** (Law and Zaniolo 2005) are known as well. Our list of online classification methods is not comprehensive; for an extended list, see the recent survey (Gaber et al. 2007).

In this chapter, we build on several surveys for online machine learning in general (Gaber et al.

2007; Bifet et al. 2011; Shalev-Shwartz et al. 2012; Fontenla-Romero et al. 2013; Gaber et al. 2014) and subfields (Widmer and Kubat 1996; Tsymbal 2004; Cheng et al. 2008; Mahdiraji 2009; Quionero-Candela et al. 2009; Kavitha and Punithavalli 2010; Žliobaite et al. 2012; Aggarwal 2013; Silva et al. 2013; Gama et al. 2014). As for other areas of machine learning, we refer to the chapters “► Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning” and “► Recommender Systems over Data Streams” in this Handbook.

Evaluation

In an infinite data stream, data available for training and testing is potentially infinite. Hence hold-out methods for selecting an independent test set seem viable at first glance and are used in a large number of online machine learning research results. However, for online evaluation, we have no control over the order in which the data is processed, and the order is not independent of the data distribution. Since the distribution generating examples and the decision models evolve over time, cross-validation and other sampling strategies are not applicable (Gama et al. 2009).

Most studies of online learning determine overall loss as a sum of losses experienced by individual training examples. Based on this so-called predictive sequential, abbreviated as **prequential** method (Dawid 1984), we define **online training and evaluation** in the following steps:

1. Based on the next unlabeled instance in the stream, we cast a prediction.
2. As soon as the true label of this instance becomes available, we assess the prediction error.
3. We update the model with the most recently observed error calculated by comparing the predicted and the true labels before proceeding with the next item of the data stream.

Prequential error is known to be a pessimistic estimator, since it may be strongly influenced

by the initial part of the sequence, when only a few examples have been processed, and the model quality is low. The effect of the beginning of the stream can be mitigated, for example, by forgetting mechanisms (Gama et al. 2009, 2013).

A further issue in online evaluation is that in reality, labels may arrive with delay (Žliobaite et al. 2012). The majority of adaptive learning algorithms require true labels to be available immediately after casting the prediction. If true labels are delayed, we have to join two streams with time delay, one for the variables and one for the labels, which can make the implementation of the prequential evaluation scheme computationally challenging.

Error metrics that can be defined for individual data points can be applied for prequential evaluation. For example, the definition of mean squared error (MSE), a popular metric for regression, is

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2, \quad (1)$$

where y_i is the actual and \hat{y}_i is the predicted class label for data point i and N is the current size of the data stream. Accuracy and error rate can be computed by a similar averaging formula.

For certain common metrics such as precision, recall, and true- and false-positive rates, the definition will involve the changing size of the set of positive, negative, or all instances, which makes the metrics difficult to interpret in a very long stream of inhomogeneous data. Although one definition of the area under the receiver operating characteristics curve (AUC) (Fogarty et al. 2005) is based on true- and false-positive rates, its online interpretation is described in Zhao et al. (2011).

Linear Models

Linear models in online machine learning date back to the perceptron algorithm (Rosenblatt 1958). The **perceptron learning** algorithm learns label y of d -dimensional input vector x as a linear combination of the coordinates,

$$\hat{y} = \mathbf{w} \cdot \mathbf{x}. \quad (2)$$

The prediction mechanism is based on an n -dimensional hyperplane of direction \mathbf{w} , which divides the instance space into two half-spaces. The margin of an example, $y \cdot \mathbf{w} \cdot \mathbf{x}$, is a signed value proportional to the distance between the instance and the hyperplane. If the margin is positive, the instance is correctly classified; otherwise, it is incorrectly classified.

The perceptron can be trained by gradient descent for hinge loss as target function. If labels y take the values ± 1 and prediction \hat{y} is defined by Eq. (2), hinge loss is equal to

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0 & \text{if } y \cdot \hat{y} \geq 1; \\ 1 - y \cdot \hat{y} & \text{otherwise,} \end{cases} \quad (3)$$

from which the gradient can be computed as follows: If prediction \hat{y} has the correct sign, and its absolute value is at least 1, that is, hinge loss is 0, then there is no change; the algorithm is passive. Otherwise, the gradient for \mathbf{w} is $-\mathbf{x} \cdot \hat{y}$, and the update rule for learning rate η is

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot \mathbf{x} \cdot \hat{y} \quad \text{if } y \cdot \hat{y} < 1. \quad (4)$$

The online gradient descent algorithm simply applies the above step to training examples in order (Langford et al. 2009). By contrast, the batch gradient descent algorithm reads the input multiple times and usually repeatedly optimizes coefficients for the same instance. Batch gradient descent can be emulated by an online algorithm: We go through training examples one by one in an online fashion and repeat multiple times over the training data (Cesa-Bianchi and Gentile 2008).

Based on the general idea of perceptron learning, several online linear models have been proposed; for a detailed overview, see Fontenla-Romero et al. (2013). The **Passive-Aggressive (PA) classifier** (Crammer et al. 2006) is a popular online linear model that works well in practice, for example, applied as the Gmail spam filter (Aberdeen et al. 2010). The main goal of the PA algorithm is to improve the convergence properties of the perceptron algorithm, the simplest

numerical optimization procedure. Several improved online optimization procedures were proposed prior to PA: Kivinen and Warmuth (1997) give an overview of numerous earlier additive and multiplicative online algorithms, all of which are solvable by gradient descent (Li and Long 2002; Gentile 2001; Crammer and Singer 2003; Kivinen et al. 2004). Based on the experiments in Crammer and Singer (2003) and Crammer et al. (2006), the Passive-Aggressive algorithm outperforms most of the earlier online linear classification methods. Perceptron learning and most of its successors apply both to classification and regression; in other words, the range of the actual label y can be both binary and continuous.

The PA algorithm solves a constrained optimization problem: We would like the new classifier to remain as close as possible to the current one while achieving at least a unit margin on the most recent example. We use the Euclidean distance of the classifiers, $\|\mathbf{w} - \mathbf{w}'\|^2$. PA optimizes for hinge loss, with the goal to keep the distance minimal and the value of the margin at least 1, that is, to solve the optimization problem

$$\mathbf{w} \leftarrow \operatorname{argmin}_{\mathbf{w}'} \|\mathbf{w} - \mathbf{w}'\|^2 \text{ s.t. } \ell(\mathbf{w}; (\mathbf{x}, y)) = 0. \quad (5)$$

The algorithm is passive if loss is 0, and otherwise aggressive, since it enforces zero loss. As shown in Crammer et al. (2006), the solution of the optimization problem yields the update rule

$$\mathbf{w} \leftarrow \mathbf{w} + \ell(\mathbf{w}; (\mathbf{x}, y)) \cdot y \cdot \mathbf{x} / \|\mathbf{x}\|^2. \quad (6)$$

In Crammer et al. (2006), variants of PA are described that introduce a slack variable $\xi \geq 0$ in Eq. (5) such that the margin must stay below ξ . Adding constant times ξ or ξ^2 to the minimization target in Eq. (5) leads to similar optimization problems. In the same paper, multi-class, cost-sensitive, and regression variants are described as well.

Although the class of linear predictors may seem restrictive, the pioneering work of Vapnik (1998) and colleagues demonstrates that by using kernels one can employ highly nonlinear

predictors as well. The **support vector machine (SVM)** learns an optimal separating hyperplane \mathbf{w}^* in a certain high-dimensional mapped space defined by the mapping $\varphi(\mathbf{x})$ over training vectors \mathbf{x} . Like with perceptron learning, the prediction has the form $\hat{y} = \mathbf{w}^* \cdot \varphi(\mathbf{x})$; however, \mathbf{w}^* may be of very high dimensionality.

Potential problems of the very high-dimensional mapped space are eliminated by the **kernel trick**. In the optimization procedure, \mathbf{w}^* turns out to be the combination of the so-called support vectors, the subset SV of training instances \mathbf{x}_i for $i \in SV$ that maximize the margin. We can obtain the parameters

$$\mathbf{w}^* = \sum_{i \in SV} \alpha_i y_i \varphi(\mathbf{x}_i), \quad (7)$$

where y_i are the labels and α_i can be found by maximizing the margin. By Eq. (7), we can reduce the prediction to computing inner products in the mapped space:

$$\mathbf{w}^* \cdot \varphi(\mathbf{x}) = \sum_{i \in SV} \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}), \quad (8)$$

The main goal of online SVM is to maintain a set of support vectors and corresponding multipliers within the limits of available memory. Whenever an online SVM learner decides to add new support vectors, others need to be discarded first, and α_i need to be updated.

To decide which support vector to discard, the definition of the span and the S-span of the support vectors defined by Vapnik and Chapelle (2000) can be used. The span of a support vector is the minimum distance of the vector from a certain set defined by all others. The S-span of a set of support vectors is the maximum span among them.

An online SVM method to update the set of support vectors from the data stream is proposed in Agarwal et al. (2008), namely, maintaining a set of support vectors and multipliers that fit into the memory limit. For a misclassified new instance, the algorithm measures the S-span of all possible ways of replacing one old support vector with the new instance and selects the best

one. Multipliers are updated by the incremental learning procedure of Cauwenberghs and Poggio (2000).

Several other, similar online SVM optimization methods are known, for example Crammer and Singer (2003), Crammer et al. (2004), Bordes and Bottou (2005), and Bordes et al. (2005). Online learning algorithms were also suggested as fast alternatives to SVM in Freund and Schapire (1999), based on the online algorithm for calculating the maximum margin linear model of Frie et al. (1998). Online gradient descent optimizers for linear models often work with kernels as well Kivinen et al. (2004).

Decision and Regression Trees

In a decision or regression tree, each internal node corresponds to a test on an attribute, while leaves contain predictors for classification or regression. To build a tree, decision tree induction algorithms iterate through each attribute and compute an information theoretic function such as entropy or Gini index for classification and variance for regression (Pang-Ning et al. 2006).

Online tree induction algorithms face the difficulty that the recursive tree construction steps cannot read past data. After a split decision is made, batch algorithms partition all data into child nodes and compute the required information theoretic function of the attributes separately in each child node. Online algorithms cannot partition past data; instead, they take advantage of the potentially infinite data and use fresh instances only in the newly created nodes.

Another problem with online tree construction is that a split decision has to be made at a certain point in time, without seeing future data. A popular solution is to use the Hoeffding criterion for a statistical guarantee that the selected split is optimal for future data as well. In the so-called Hoeffding tree or very fast decision tree (VFDT) (Domingos and Hulten 2000), attribute information theoretic functions are maintained over the stream. If the Hoeffding criterion is met, a split decision is made, and the attribute statistics over the new child nodes are computed based on

the new data from the stream. Similar methods for regression trees are described in Alberg et al. (2012) and Ikonomovska et al. (2015). For online vertical parallel distributed Hoeffding trees, see Kourtellis et al. (2016).

One problem in decision tree construction on streaming data is the cost of maintaining attribute statistics for many-valued attributes. For such attributes, a low granularity histogram has to be maintained. In Jin and Agrawal (2003), a method is described that partitions the range of a numerical attribute into intervals and uses statistical tests to prune these intervals.

Another difficulty is caused by nonstationary data, since each new child node is processed based on a new portion of data from the stream. Decision trees over evolving streams are considered in Bifet and Gavaldà (2009), Bifet (2010), and Bifet et al. (2017). We give an overview of general methods for nonstationary data in the chapter “► Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning” of this Handbook.

Ensemble Methods

Ensemble methods build multiple, potentially different models on potentially different subsets of instances and attributes (Pang-Ning et al. 2006). The simplest example is **bagging** where several base models are trained based on samples with replacement. Sampling with replacement can be simulated online (Oza 2005; Bifet et al. 2010). A special ensemble technique, the online random forest algorithm, is described in Denil et al. (2013).

A highly successful ensemble technique is **boosting**, in which we generate a sequence of base models by incorporating the prediction error of the previous models when constructing the next. For example, in AdaBoost, an algorithm designed for online learning in its first description (Freund and Schapire 1995), the next classifier is trained by weighting instances based on the function of the error of previous classifiers. In gradient boosting (Chen and Guestrin 2012), the

next model is trained on the residual, that is, the difference of the training label and the continuous predicted label of the previous classifiers.

For online boosting, the difference compared to batch boosting is that the performance of a base model cannot be observed on the entire training set, only on the examples seen earlier. Like with online decision tree induction, decisions need to be taken based only on part of the training data. Various online boosting algorithms are described in Oza (2005), Chen et al. (2012), and Beygelzimer et al. (2015), based on the ideas of parallel boosting via approximation Fan et al. (1999), Palit and Reddy (2012), and Lazarevic and Obradovic (2002). For gradient boosted trees (Chen and Guestrin 2012), a recent, most successful classification method, the online version, is described in Vasiloudis et al. (2017).

Bayes Models

Bayesian networks were one of the earliest applications for online learning (Friedman and Goldszmidt 1997), with the first methods mostly using mini-batch updates (Buntine 1991). Bayesian learning methods maintain conditional probability tables $P(\mathbf{x}|y)$ by counting, where \mathbf{x} is a feature vector and y is its label. Considering the conditional probability tables and the class distribution as priors, the predicted class of an instance will be the one that maximizes the posterior probability computed by the Bayes rule (Pang-Ning et al. 2006).

The simplest, naive Bayes model makes the “naive” assumption that each input variable is conditionally independent given the class label (Duda et al. 2012). While this model works surprisingly well (Domingos and Pazzani 1997), a weaker assumption is needed in Bayesian networks (Pearl 2014), in which we represent each variable with a node in a directed acyclic graph. Rather than assuming independence naively, we assume that each variable is conditionally independent given its parents in the graph.

For online learning, it is easy to update the conditional probabilities both for naive Bayes and for Bayesian networks (Friedman and Goldszmidt 1997), provided that they fit into internal memory. Methods for updating the network structure online are described, for example, in Friedman and Goldszmidt (1997) and Chen et al. (2001).

Neural Networks

Neural networks and deep learning have shown great promise in many practical applications ranging from speech recognition (Hinton et al. 2012) and visual object recognition (Krizhevsky et al. 2012) to text processing (Collobert and Weston 2008). One of the earliest applications of online trained neural networks is EEG classification (Haselsteiner and Pfurtscheller 2000).

Gradient descent is perhaps the most commonly used optimization procedure for training neural networks (LeCun et al. 1998), which naturally leads to online learning algorithms (Juang and Lin 1998) as well. The traditional formulation of gradient descent is impractical for very large neural networks. A scalable online distributed version, Downpour SGD (Dean et al. 2012), uses asynchronous parameter updates in conjunction with a parameter server described in the chapter “► [Overview of Online Machine Learning in Big Data Streams](#)” of this Handbook.

Cross-References

- [Overview of Online Machine Learning in Big Data Streams](#)
- [Recommender Systems over Data Streams](#)
- [Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#)

Acknowledgements Support from the EU H2020 grant Streamline No 688191 and the “Big Data—Momentum” grant of the Hungarian Academy of Sciences.

References

- Aberdeen D, Pacovsky O, Slater A (2010) The learning behind gmail priority inbox. In: LCCC: NIPS 2010 workshop on learning on cores, clusters and clouds
- Agarwal S, Saradhi VV, Karnick H (2008) Kernel-based online machine learning and support vector reduction. *Neurocomputing* 71(7):1230–1237
- Aggarwal CC (2013) A survey of stream clustering algorithms. In: Aggarwal CC, Reddy CK (eds) *Data clustering: algorithms and applications*. CRC press, Boca Raton, p 231
- Alberg D, Last M, Kandel A (2012) Knowledge discovery in data streams with regression tree methods. *Wiley Interdisciplinary Rev Data Min Knowl Discov* 2(1): 69–78
- Beygelzimer A, Kale S, Luo H (2015) Optimal and adaptive algorithms for online boosting. In: Proceedings of the 32nd international conference on machine learning (ICML-15), pp 2323–2331
- Bifet A (2010) Adaptive stream mining: pattern learning and mining from evolving data streams. In: Proceedings of the 2010 conference on adaptive stream mining: pattern learning and mining from evolving data streams. Ios Press, pp 1–212
- Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: International symposium on intelligent data analysis. Springer, pp 249–260
- Bifet A, Holmes G, Pfahringer B (2010) Leveraging bagging for evolving data streams. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, Berlin/Heidelberg
- Bifet A, Kirkby R, Pfahringer B (2011) Data stream mining: a practical approach. Technical report, University of Waikato
- Bifet A, Zhang J, Fan W, He C, Zhang J, Qian J, Holmes G, Pfahringer B (2017) Extremely fast decision tree mining for evolving data streams. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1733–1742
- Bordes A, Bottou L (2005) The huller: a simple and efficient online SVM. In: ECML. Springer, pp 505–512
- Bordes A, Ertekin S, Weston J, Bottou L (2005) Fast kernel classifiers with online and active learning. *J Mach Learn Res* 6:1579–1619
- Buntine W (1991) Theory refinement on Bayesian networks. In: Proceedings of the seventh conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., pp 52–60
- Cauwenberghs G, Poggio T (2000). Incremental and decremental support vector machine learning. In: Proceedings of the 13th international conference on neural information processing systems (NIPS’00), Denver. MIT Press, Cambridge, pp 388–394
- Cesa-Bianchi N, Gentile C (2008) Improved risk tail bounds for on-line algorithms. *IEEE Trans Inf Theory* 54(1):386–390

- Chen R, Sivakumar K, Kargupta H (2001) An approach to online Bayesian learning from multiple data streams. In: Workshop on ubiquitous data mining for mobile and distributed environments, held in conjunction with joint 12th European conference on machine learning (ECML'01) and 5th European conference on principles and practice of knowledge discovery in databases (PKDD'01), Freiburg
- Chen ST, Lin HT, Lu CJ (2012) An online boosting algorithm with theoretical justifications. In: Proceedings of the 29th international conference on machine learning. Omnipress, pp 1873–1880
- Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 785–794
- Cheng J, Ke Y, Ng W (2008) A survey on algorithms for mining frequent itemsets over data streams. *Knowl Inf Syst* 16(1):1–27
- Collobert R, Weston J (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the 25th international conference on machine learning. ACM, pp 160–167
- Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y (2006) Online passive-aggressive algorithms. *J Mach Learn Res* 7:551–585
- Crammer K, Kandola J, Singer Y (2004) Online classification on a budget. In: Advances in neural information processing systems, pp 225–232
- Crammer K, Singer Y (2003) Ultraconservative online algorithms for multiclass problems. *J Mach Learn Res* 3(Jan):951–991
- Dawid AP (1984) Present position and potential developments: some personal views: statistical theory: the prequential approach. *J R Stat Soc Ser A (General)* 147(2):278–292
- Dean J, Corrado G, Monga R, Chen K, Devin M, Mao M, Senior A, Tucker P, Yang K, Le QV et al (2012) Large scale distributed deep networks. In: Advances in neural information processing systems. Curran Associates, Inc., New York, pp 1223–1231
- Denil M, Matheson D, Nando D (2013) Consistency of online random forests. In: Proceedings of the 30th international conference on machine learning (ICML'13), pp 1256–1264
- Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80
- Domingos P, Pazzani M (1997) On the optimality of the simple Bayesian classifier under zero-one loss. *Mach Learn* 29(2):103–130
- Duda RO, Hart PE, Stork DG (2012) Pattern classification. John Wiley & Sons, Somerset
- Fan W, Stolfo SJ, Zhang J (1999) The application of adaboost for distributed, scalable and on-line learning. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 362–366
- Fogarty J, Baker RS, Hudson SE (2005) Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction. In: Proceedings of graphics interface 2005, GI'05. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, pp 129–136. <http://portal.acm.org/citation.cfm?id=1089508.1089530>
- Fontenla-Romero Ó, Guijarro-Berdiñas B, Martínez-Rego D, Pérez-Sánchez B, Peteiro-Barral D (2013) Online machine learning. Efficiency Scalability Methods Comput Intellect 27:27–54
- Freund Y, Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: European conference on computational learning theory. Springer, pp 23–37
- Freund Y, Schapire RE (1999) Large margin classification using the perceptron algorithm. *Mach Learn* 37(3):277–296
- Frie TT, Cristianini N, Campbell C (1998) The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines. In: Machine learning: proceedings of the fifteenth international conference (ICML'98), pp 188–196
- Friedman J, Hastie T, Tibshirani R (2001) The elements of statistical learning. Springer series in statistics, vol 1. Springer, New York
- Friedman N, Goldszmidt M (1997) Sequential update of Bayesian network structure. In: Proceedings of the thirteenth conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., pp 165–174
- Gaber M, Zaslavsky A, Krishnaswamy S (2007) A survey of classification methods in data streams. In: Data streams. Springer, Boston, pp 39–59
- Gaber MM, Gama J, Krishnaswamy S, Gomes JB, Stahl F (2014) Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdisciplinary Rev Data Min Knowl Discov* 4(2):116–138
- Gama J, Sebastião R, Rodrigues PP (2009) Issues in evaluation of stream learning algorithms. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 329–338
- Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. *Mach Learn* 90(3): 317–346
- Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv (CSUR)* 46(4):44
- Gentile C (2001) A new approximate maximal margin classification algorithm. *J Mach Learn Res* 2(Dec):213–242
- Haselsteiner E, Pfurtscheller G (2000) Using time-dependent neural networks for eeg classification. *IEEE Trans Rehabil Eng* 8(4):457–463
- Hinton G, Deng L, Yu D, Dahl GE, Mohamed Ar, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN et al (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process Magazine* 29(6):82–97

- Ikonomovska E, Gama J, Džeroski S (2015) Online tree-based ensembles and option trees for regression on evolving data streams. *Neurocomputing* 150:458–470
- Jin R, Agrawal G (2003) Efficient decision tree construction on streaming data. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 571–576
- Juang CF, Lin CT (1998) An online self-constructing neural fuzzy inference network and its applications. *IEEE Trans Fuzzy Syst* 6(1):12–32
- Kavitha V, Punithavalli M (2010) Clustering time series data stream-a literature survey. arXiv preprint arXiv:1005.4270
- Kivinen J, Smola AJ, Williamson RC (2004) Online learning with kernels. *IEEE Trans Signal Process* 52(8):2165–2176
- Kivinen J, Warmuth MK (1997) Exponentiated gradient versus gradient descent for linear predictors. *Inf Comput* 132(1):1–63
- Kourtellis N, Morales GDF, Bifet A, Murdopo A (2016) Vht: vertical hoeffding tree. In: 2016 IEEE international conference on big data (Big Data). IEEE, pp 915–922
- Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems (NIPS'12), Lake Tahoe, vol 1. Curran Associates Inc., pp 1097–1105
- Langford J, Li L, Zhang T (2009) Sparse online learning via truncated gradient. *J Mach Learn Res* 10:777–801
- Law YN, Zaniolo C (2005) An adaptive nearest neighbor classification algorithm for data streams. In: European conference on principles of data mining and knowledge discovery. Springer, pp 108–120
- Lazarevic A, Obradovic Z (2002) Boosting algorithms for parallel and distributed learning. *Distrib Parallel Databases* 11(2):203–229
- LeCun Y, Bottou L, Orr GB, Müller KR (1998) Efficient backprop. In: Orr G, Müller K-R (eds) Neural networks: tricks of the trade. Springer, Berlin/New York, pp 9–50
- Li Y, Long PM (2002) The relaxed online maximum margin algorithm. *Mach Learn* 1(46):361–387
- Mahdiraji AR (2009) Clustering data stream: a survey of algorithms. *Int J Knowl-Based Intell Eng Syst* 13(2):39–44
- Obermaier B, Guger C, Neuper C, Pfurtscheller G (2001) Hidden Markov models for online classification of single trial eeg data. *Pattern Recognit Lett* 22(12):1299–1309
- Oza NC (2005) Online bagging and boosting. In: 2005 IEEE international conference on systems, man and cybernetics, vol 3. IEEE, pp 2340–2345
- Palit I, Reddy CK (2012) Scalable and parallel boosting with MapReduce. *IEEE Trans Knowl Data Eng* 24(10):1904–1916
- Pang-Ning T, Steinbach M, Kumar V et al (2006) Data mining cluster analysis: basic concepts and algorithms. In: Introduction to data mining. Pearson Addison Wesley, Boston/Toronto
- Pearl J (2014) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco
- Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2009) Dataset shift in machine learning. The MIT Press, Cambridge
- Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol Rev* 65(6):386
- Shalev-Shwartz S et al (2012) Online learning and online convex optimization. *Found Trends® Mach Learn* 4(2):107–194
- Silva JA, Faria ER, Barros RC, Hruschka ER, de Carvalho AC, Gama J (2013) Data stream clustering: a survey. *ACM Comput Surv (CSUR)* 46(1):13
- Tsymbal A (2004) The problem of concept drift: definitions and related work. Technical Report 2. Computer Science Department, Trinity College, Dublin
- Vapnik V, Chapelle O (2000) Bounds on error expectation for support vector machines. *Neural Comput* 12(9):2013–2036
- Vapnik VN (1998) Statistical learning theory, vol 1. Wiley, New York
- Vasiloudis T, Beligianni F, Morales GDF (2017) Boost-vht: boosting distributed streaming decision trees. In: CIKM
- Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. *Mach Learn* 23(1):69–101
- Zhao P, Jin R, Yang T, Hoi SC (2011) Online AUC maximization. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp 233–240
- Žliobaite I, Bifet A, Gaber M, Gabrys B, Gama J, Minku L, Musial K: Next challenges for adaptive learning systems. *ACM SIGKDD Explor News* 14(1), 48–55 (2012)

Online Machine Learning in Big Data Streams: Overview

András A. Benczúr¹, Levente Kocsis¹, and Róbert Pálavics²

¹Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary

²Department of Computer Science, Stanford University, Stanford, CA, USA

Synonyms

Incremental learning; Online machine learning on streams; Stream learning

Definitions

- Data stream algorithms process a continuous stream of data with only a limited possibility to store past records.
- Online machine learning covers methods that update their models after observing a new event and can immediately serve predictions based on the updated model.

Overview

In this chapter, we investigate online learning based recommender algorithms that can efficiently handle nonstationary datasets. We show that online learning for recommendation is rather usual than the exceptional task: For example, if no user history is available, we have to build a user model on the fly, based on the interactions in the live user session.

To the best of our knowledge, this is the first survey with a comprehensive overview of the ideas for recommendation over streaming data and their implementation in various distributed data stream processing systems.

The chapter is based on the notions of online learning, as introduced in the chapter “Overview of Online Machine Learning in Big Data Streams” of this Encyclopedia.

Introduction

Big data analytics promise to deliver valuable business insights. However, this is difficult to realize using today’s state-of-the-art technologies, given the flood of data generated from various sources. A few years ago, the term **fast data** (Lam et al. 2012) arose to capture the idea that **streams** of data are generated at very high rates and that these need to be analyzed quickly in order to arrive at actionable intelligence.

Fast data can flood from network measurements, call records, web page visits, sensor readings, and so on (Bifet et al. 2011). The fact that such data arrives continuously in multiple, rapid, time-varying, possibly

unpredictable, and unbounded streams appears to yield some fundamentally new research problems. Examples of such applications include financial applications (Zhu and Shasha 2002), network monitoring (Babu and Widom 2001; Abadi et al. 2003), security, sensor networks (De Francisci Morales et al. 2016), Twitter analysis (Bifet and Frank 2010), and more (Fontenla-Romero et al. 2013).

Traditional data processing assumes that data is available for multiple access, even if in some cases it resides on disk and can only be processed in larger chunks. In this case, we say that the data is **at rest**, and we can perform **batch processing**. Database systems, for example, store large collections of data and allow users to initiate queries and transactions.

Fast data or **data in motion** is closely connected to and in certain cases used as a synonym of the **data stream** computational model (Muthukrishnan et al. 2005). In this model, data arrives continuously in a potentially infinite stream that has to be processed by a resource-constrained system. The main restriction is that the main memory is small and can contain only a small portion of the stream; hence most of the data has to be immediately discarded after processing.

In one of the earliest papers that describe a system for data stream processing (Abadi et al. 2003), the needs of monitoring applications are described. The tasks relevant for monitoring applications differ from conventional processing of data at rest in that the software system must process and react to continuous input from multiple sources. The authors introduce the data active, human passive model, in which the system permanently processes data to provide alerts for humans.

Several surveys for online machine learning in general (Gaber et al. 2007, 2014; Bifet et al. 2011; Shalev-Shwartz et al. 2012; Fontenla-Romero et al. 2013) and subfields (Widmer and Kubat 1996; Tsymbal 2004; Cheng et al. 2008; Mahdiraji 2009; Quionero-Candela et al. 2009; Kavitha and Punithavalli 2010; Žliobaite et al. 2012; Aggarwal 2013; Silva et al. 2013; Gama et al. 2014) have appeared recently. Our survey

is different, first of all, in that we focus on three aspects: the data stream computational model, the adaptive methods for handling concept drift, and the distributed software architecture solutions for streaming. We also elaborate on systems for machine learning by distributed data stream processing. In particular, we explore the idea of using parameter servers.

In this chapter, we give an overview of distributed software architectures for online machine learning. The handbook contains three related chapters that describe specific areas of online machine learning. Online classification and regression is explained in “[Online Machine Learning Algorithms Over Data Streams](#),” recommendation in “[Recommender Systems Over Data Streams](#),” and finally additional topics in “[Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#).”

Requirements for Machine Learning Over Fast Data Streams

Needs and opportunities for **machine learning over fast data streams** are stimulated by a rapidly growing number of industrial, transactional, sensor, and other applications (Žliobaite et al. 2012). The concept of online machine learning is summarized in one of the earliest overviews of the field (Widmer and Kubat 1996). In the data stream computational model, only a small portion of the data can be kept available for immediate analysis (Henzinger et al. 1998). This has both algorithmic and statistical consequences for machine learning: Suboptimal decisions on earlier parts of the data may be difficult to unwind and, if needed, require low memory sampling and summarization procedures. For data streaming applications, incremental or online learning fits best.

Requirement 1 Online learning updates its model after each data instance without access to all past data; hence the constraints of the data streaming computational model apply.

Data streaming is not just a technical restriction on machine learning: Fast data is

not just about processing power but also about fast semantics. Large databases available for mining today have been gathered over months or years, and the underlying processes generating them have changed during this time, sometimes radically (Hulten et al. 2001). In data analysis tasks, fundamental properties of the data may change quickly, which makes gradual manual model adjustment procedures inefficient and even infeasible (Žliobaite et al. 2012). Traditional, batch learners build static models from finite, static, identically distributed data sets. By contrast, stream learners need to build models that evolve over time. Processing will strongly depend on the order of examples generated from a continuous, nonstationary flow of data. Modeling is hence affected by potential concept drifts or changes in distribution (Gama et al. 2013).

Requirement 2 Adaptive machine learning models are needed to handle concept drift.

As an additional consequence of Requirement 2, adaptive learning also affects the way evaluation is performed. Potentially in every unit of time, the system may return predictions from different models, and we may receive too few predictions from a particular model to evaluate by traditional metrics. Instead, we have to define error measures that we can minimize in a feedback system: Predictions are made for a stream of objects one by one, and the correct answer is received immediately afterward. A discrepancy between the prediction and the observed value serves as feedback, which may immediately trigger modifications to the model (Widmer and Kubat 1996).

Finally, the third important aspect for online learning from big data is algorithmic. In order to cope with the volume of the data, processing has to be distributed. Clusters of machines are hard to manage, and hardware failure must be mitigated in the case of an application running on thousands of servers. Map-Reduce (Dean and Ghemawat 2008) was the first programming abstraction designed to manage the cluster, provide fault tolerance, and ease software development and

debugging. While Map-Reduce is designed for batch processing, distributed data stream processing needs other solutions, such as communication between processing elements (Neumeyer et al. 2010) via an interconnection topology (Toshniwal et al. 2014). For an outlook, mostly batch distributed data mining solutions are surveyed in Fan and Bifet (2013).

Requirement 3 Online learning from big data has to be implemented in a distributed stream processing architecture.

Data-intensive applications often work in the data stream computational model (Babcock et al. 2002), in which the data is transient: Some or all of the input data is not available for random access from disk or memory. The data elements in the stream arrive online and can be read at most once; in case of a failure, it is possible that the data elements cannot be read at all. The system has no control over the order in which data elements arrive to be processed either within a data stream or across data streams.

The strongest constraint for processing data streams is the fact that once an element from a data stream has been processed, it has to be discarded or archived. Only selected past data elements can be accessed by storing them in memory, which is typically small relative to the size of the data streams. Many of the usual data processing operations would need random access to the data (Babcock et al. 2002): For example, only a subset of SQL queries can be served from the data stream. As surveyed in Muthukrishnan et al. (2005), data stream algorithms can tackle this constraint by a variety of strategies, including adaptive sampling in sliding windows, selecting representative distinct elements, and summarizing data in low memory data structures, also known as sketches or synopses.

When designing online machine learning algorithms, we have to take several algorithmic and statistical considerations into account. The first problem we face is the restrictions of the computational model. As we cannot store all the input, we cannot unwind a decision made on past data. For example, we can use statistical tests to choose from competing hypotheses (Domingos

and Hulten 2000), which give theoretical guarantees in the case of identically distributed data.

A second problem with real stream learning tasks is that data typically changes over time, for example, due to concept drifts (Widmer and Kubat 1996). For changing distributions, we can even use the streaming computational model to our advantage: By permanent training, we can adapt to concept drift by overwriting model parameters based on insights from fresh data and thus forgetting the old distribution (Frigó et al. 2017). Possible means of concept drift adaptation, however, will depend on the task, the data, and the choice of the algorithm (Widmer and Kubat 1996; Klinkenberg and Joachims 2000).

In this chapter, we give a brief overview of how different, mostly open source, software projects manage machine learning tasks over streaming data. We note that very active, ongoing research in the field may make parts of our description obsolete very quickly. First, in section “[Data Stream Processing Engines](#),” we provide a summary of the most important distributed data stream processing engines with active development at the time of writing. Next, in section “[Taxonomy of Machine Learning Tools](#),” we give a taxonomy of the possible machine learning solutions with respect to support from streaming data and distributed processing. Since operation on shared-nothing distributed architectures is a key requirement for big data processing solutions, we discuss the main machine learning parallelization strategies in section “[Parallel Learning and the Parameter Server](#).“ One particularly popular solution, the parameter server is also described in more detail. Finally, in section “[Stream Learning Libraries](#),” we list functionalities of stream learning libraries as of the time of writing.

Data Stream Processing Engines

Distributed stream learning libraries are usually either part of **data stream processing engines (DSPEs)** or built on top of them through interfaces. For a better understanding of the software architecture, we give an overview of DSPEs next.

Since the emergence of early systems such as Aurora (Abadi et al. 2003; Arasu et al. 2003), several DSPEs have been developed. For a survey of DSPEs, see Ranjan (2014). The main focus of the most recent evolving DSPEs is to provide simplicity, scalability, stateful processing, and fault tolerance with fast recovery. In terms of simplicity, the most important goal is to overcome the need to integrate the DSPE with a batch processing engine as in the so-called lambda architecture described, among other places, in Marz and Warren (2015) and Kiran et al. (2015).

In this section, we focus on Apache Spark (Zaharia et al. 2010) and Apache Flink (Carbone et al. 2015), since at the time of writing, these DSPEs have the most active development for learning from streams. Other systems usually provide machine learning functionalities by interfacing with SparkML, a Spark-based machine learning library, or SAMOA (Morales and Bifet 2015), a stream learning library designed to work as a layer on top of general DSPEs. We list the functionality of these systems in section “[Stream Learning Libraries](#).”

Without attempting to be exhaustive, we list some other main DSPEs. Note that this field is very active, and several systems with large impact on both DSPEs and stream learning have already stopped development. For example, several active projects have borrowed concepts from the **S4 project** (Neumeyer et al. 2010), which retired in 2014. Proprietary systems are summarized in Gualtieri et al. (2013); it appears to be the case that commercial developers as of yet have no special focus on data stream processing; hence our main goal is to give an overview of the open source solutions.

Storm (Toshniwal et al. 2014) relies on the concept of the connection topology of the processing elements. The topology is allowed to contain cycles; however, in case of a failure with cycles, neither the exactly-once nor the at-least-once processing condition can be enforced. **Samza** (Noghabi et al. 2017) provides a design for fast recovery after failures independent of the state size. As a new advantage, stateful streaming systems have the possibility to emulate batch operation; hence the need for the lambda

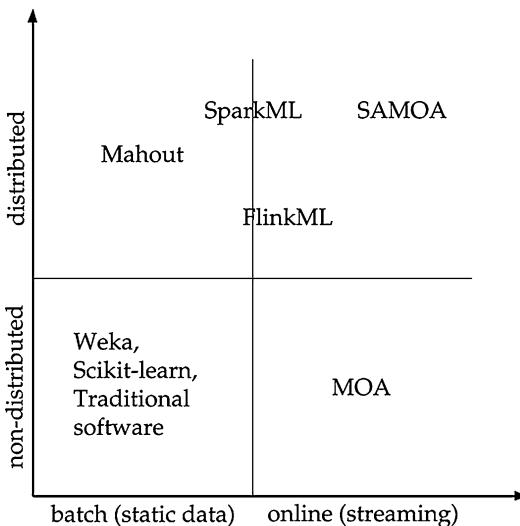
architecture can be eliminated. **Beam** (Akidau et al. 2015), based on the Google Cloud dataflow model (Chambers et al. 2010; Akidau et al. 2013, 2015), focuses on event time windows to process out-of-order data. Beam can be connected to the deep learning framework TensorFlow (Dean et al. 2012) for batch machine learning. Finally, while most of the above systems are distributed, we mention Esper (Bernhardt and Vasseur 2007) as a prominent single machine, in-memory DSPE.

One DSPE in the focus of this article, **Spark** (Zaharia et al. 2010) treats stream processing as a sequence of small batch computations. Records in the stream are collected into micro-batches (by time), and a short-lived batch job is scheduled to process each micro-batch. The advantage of this approach is its intuitive transition from batch to streaming with straightforward fault tolerance and interaction with batch programs. The main disadvantage is higher latency due to the inherent scheduling overhead for the micro-batch.

Another DSPE of our choice, **Flink** (Carbone et al. 2015) provides consistent managed state with exactly-once guarantees while achieving high throughput and low latency, serving both batch and streaming tasks. Flink handles the stream event by event in a true streaming fashion through the underlying streaming (dataflow) runtime model. While this provides more fine-grained access to the stream, it does not come with a throughput overhead due to various runtime optimizations such as buffering of output records. The advantages are very low processing latency and a natural stateful computational model. The disadvantages are that fault tolerance and load balancing are more challenging to implement. Flink is primarily for stream processing. Flink batch tasks can be expressed by using loops in stream processing, as we will see in the next section.

Taxonomy of Machine Learning Tools

In this article, we survey online machine learning tools for big data. Our main focus are models, architectures, and software libraries that process



Online Machine Learning in Big Data Streams: Overview, Fig. 1 Taxonomy of machine learning tools

streaming data over distributed, shared-nothing architectures. As shown in Fig. 1, the two key distinguishing features of machine learning tools are whether they are distributed and whether they are based on static or streaming data.

As the least restrictive of the four quadrants in Fig. 1, **batch non-distributed** tools can implement any method from the other quadrants. If scale permits, data streams can first be stored and then analyzed at rest, and distributed processing steps can be unfolded to run sequentially on a single-processor system. Traditional machine learning tools, for example, R, Weka, and scikit-learn, fall into this quadrant.

Distributed batch machine learning systems (Fan and Bifet 2013) typically implement algorithms by using the Map-Reduce principle (Dean and Ghemawat 2008). Perhaps the best-known system is Mahout (Owen et al. 2011) built on top of Hadoop (White 2010), but a very rich variety of solutions exists in this field. GraphLab (Low et al. 2012), withdrawn from the market in 2016, was another mostly batch tool that has also influenced online systems.

Online learning solutions cover algorithms that immediately build models after seeing a relatively small portion of the data. By this require-

ment, we face the difficulty of not necessarily being able to undo a suboptimal decision made in an earlier stage, based on data that is no longer available for the algorithm.

The first library for online machine learning, MOA (Bifet et al. 2010) collects a variety of models suitable for training online, most of which we describe in the “[Online Machine Learning Algorithms Over Data Streams](#)” chapter of this handbook. Based on MOA concepts, SAMOA (Morales and Bifet 2015) is a distributed framework – a special purpose DSPE – and library that provides distributed implementation for most MOA algorithms. Online learning recommender systems, both distributed and nondistributed, are also described in Pálovics et al. (2017). Another recent nondistributed online learning tool is described in Bifet et al. (2017).

For a **distributed online learning** software architecture, the underlying system needs to be a DSPE. In addition to SAMOA, which can be considered a DSPE itself, most DSPEs of the previous section can be used for distributed online learning. For example, Flink, Samza, and Storm implement interfaces to use SAMOA libraries. On the other hand, a DSPE can also implement batch machine learning algorithms: For example, the machine learning library SparkML is mostly batch and FlinkML is partly batch.

Combined batch and online machine learning solutions are of high practical relevance. We can train our model batch based on a precompiled sample and then apply prediction for a live data stream. DSPE solutions are progressing in this area. One recent result, Clipper is a low-latency online prediction system (Crankshaw et al. 2017) with a model abstraction layer that makes it easy to serve pre-trained models based on a large variety of machine learning frameworks.

Another batch approach to learning from time-changing data is to repeatedly apply a traditional learner to a sliding window of examples: As new examples arrive, they are inserted into the beginning of the window. Next, a corresponding number of examples are removed from the end of the window, and the learner is reapplied, as in early research on concept drift in learning

from continuous data (Widmer and Kubat 1996; Hulten et al. 2001). Finally, as a combination with online learning methods, the batch-trained model can be incrementally updated by a streaming algorithm (Frigó et al. 2017).

Parallel Learning and the Parameter Server

In order to design distributed modeling algorithms, we have to elaborate on parallelization strategies. **Horizontal** or **data parallel** systems partition the training data and build separate models on subsets that may eventually get merged. Such a solution is applied, for example, for training XGBoost trees (Chen et al. 2016). This approach, however, will typically depend on the partitioning and lead to heuristic or approximate solutions.

Vertical parallel or model parallel training

solutions partition across attributes in the same data point, rather than partitioning the training data coming from the stream. Each training point is split into its constituting attributes, and each attribute is sent to a different processing element. For example, for linear models trained by gradient descent, coefficients can be stored and updated by accessing a distributed store (Li et al. 2014a). As another example, the fitness of attributes for a split in a decision tree construction can be computed in parallel (Morales and Bifet 2015). Further examples such as TensorFlow (Dean et al. 2012), Petuum (Xing et al. 2015), and MXNet (Chen et al. 2015) are also capable of model parallel training. The drawback is that in order to achieve good performance, there must be sufficient inherent parallelism in the modeling approach.

The **parameter server** introduced in Smola and Narayananurthy (2010) is a popular way to implement model parallel training, with several variants developed (Ho et al. 2013; Li et al. 2013, 2014a,b), including an application for deep learning (Dean et al. 2012). The main idea is to simplify the development of distributed machine learning applications by allowing access to shared parameters as key–value pairs. As shown

in Fig. 2, the compute elements are split into two subsets: Parameters are distributed across a group of **server nodes**, while data processing and computation are performed at **worker nodes**. Any node can both push out its local parameters and pull parameters from remote nodes. Parallel tasks are typically asynchronous, but the algorithm designer can be flexible in choosing a consistency model.

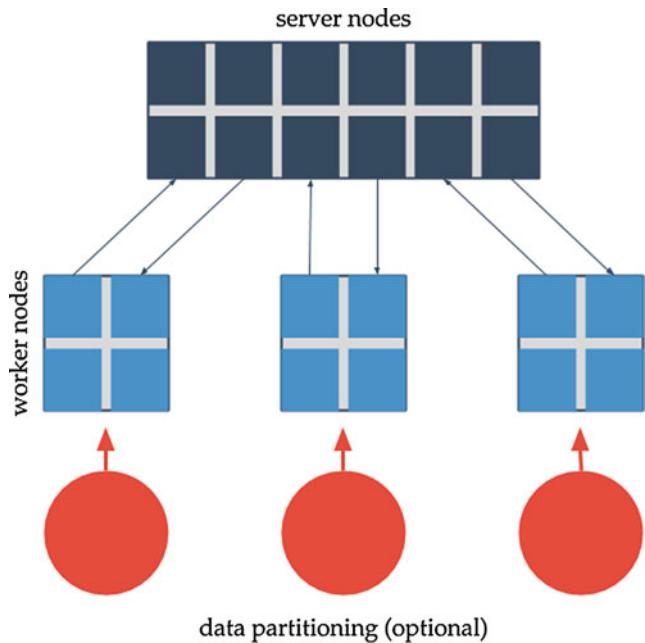
Various parameter server systems are summarized best in Li et al. (2014a). The first generation (Smola and Narayananurthy 2010) uses distributed key-value stores such as Memcached (Fitzpatrick 2004) to store the parameters. More recent solutions (Dean et al. 2012; Li et al. 2013) implement specific parameter access interfaces. For example, as the first step toward a general platform, (Ho et al. 2013) design a table-based interface for stale synchronous operation, which they use to implement a wide variety of applications.

Most results (Smola and Narayananurthy 2010; Ho et al. 2013; Li et al. 2014a) describe parameter servers for variants of regression as well as unsupervised modeling by latent Dirichlet allocation. Another popular use is classification by deep neural networks (Dean et al. 2012; Li et al. 2013). An implementation for Multiple Additive Regression Trees is given in Zhou et al. (2017).

Recommendation algorithms can also be implemented by the parameter server principle. Batch implementations of asynchronous distributed stochastic gradient descent recommender algorithms are given in Ho et al. (2013) and Schelter et al. (2014). A recent comparison of distributed recommenders, including an online one based on a Flink parameter server, is given in Pálovics et al. (2017).

The parameter server idea fits the data stream as well as the batch computational models. For example, if it suffices to read the data only once and in input order for gradient descent, a batch parameter server such as in Li et al. (2014a) immediately yields a streaming algorithm. Yet, most applications are for batch modeling only (Smola and Narayananurthy 2010; Ho et al. 2013; Li et al. 2014a; Schelter et al. 2014; Zhou et al.

Online Machine Learning in Big Data Streams: Overview,
Fig. 2 The parameter server architecture for distributed machine learning



2017). As examples of parameter servers for online learning, applications for reinforcement learning can be found in Nair et al. (2015) and for recommenders in Pálovics et al. (2017).

Stream Learning Libraries

We provide an overview of the present machine learning functionalities of the main open source engines. Note that all these engines are under active development with a very large number of components already available as research prototypes or pull requests. For this reason, we only want to give a representative overview of the main components.

MOA and SAMOA

SAMOA (Morales and Bifet 2015), a distributed online learning library, is based on the concepts of MOA (Bifet et al. 2010), a single machine library. SAMOA provides model parallel implementations of specific algorithms by using the concepts of Storm (Toshniwal et al. 2014): Processing elements are connected in a loop-free topology, which connects the various pieces of user code. SAMOA can run on top of a DSPE

with a flexible interface connection. SAMOA connectors are implemented for all active DSPEs in section “Data Stream Processing Engines.”

For classification, SAMOA (De Francisci Morales 2013; Morales and Bifet 2015) provides the Vertical Hoeffding Tree (VHT), a distributed version of a streaming decision tree (Domingos and Hulten 2000). For clustering, it includes an algorithm based on CluStream (Aggarwal et al. 2003). The library also includes meta-algorithms such as bagging and boosting.

Apache Spark

Spark has a very rich set of batch machine learning functionalities, including linear, tree, support vector machine, and neural network models for classification and regression, ensemble methods as well as explicit and implicit alternating least squares for recommender systems, and many more listed at <https://spark.apache.org/mllib/>. However, the only streaming algorithm in Spark MLLib is a linear model with ongoing work regarding online recommender and latent Dirichlet allocation prototypes. Spark has no SAMOA connector yet.

Spark has several parameter server implementations, most of which are batch only. We men-

tion two projects with recent activity. Glint is a parameter server-based latent Dirichlet allocation implementation, which is described in Jagerman et al. (2017). Angel is a general parameter server (Jiang et al. 2017) that has implementations for logistic regression, SVM, matrix factorization, latent Dirichlet allocation, and more. Angel supports synchronous, asynchronous, and stale synchronous processing (Jiang et al. 2017).

Apache Flink

Flink has a loosely integrated set of machine learning components, most of which are collected at <https://github.com/FlinkML>. In addition to the SAMOA connector at <https://github.com/apache/incubator-samoa/tree/master/samoa-flink>, Flink has a true streaming parameter server implementation at <https://github.com/FlinkML/flink-parameter-server>, which includes a Passive Aggressive classifier and gradient descent matrix factorization. Finally, Flink can serve online predictions by models trained on any system supporting the PMML standard (Grossman et al. 2002), using the JPMMML library at <https://github.com/FlinkML/flink-jpmmml>.

In Flink, the parameter server is implemented as part of the data stream API at <https://github.com/FlinkML/flink-parameter-server>. Since the communication between workers and servers is two-way, the implementation involves loops in stream processing. As mentioned in section “[Data Stream Processing Engines](#),” exactly-once processing and fault tolerance are conceptually difficult, and implementation is not yet complete as of the time of writing (Carbone et al. 2017).

Conclusions

In this chapter, we provided an overview of the general requirements for online machine learning. Some algorithms are described in more details in three more chapters of this handbook, including “[Online Machine Learning Algorithms Over Data Streams](#)”; “[Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#)”; and “[Recommender Systems Over Data Streams](#)”.

This article is a reference material and not a survey. We do not attempt to be comprehensive in describing all existing methods and solutions; rather, we give pointers to the most important resources in the field. All related subfields, online algorithms, online learning, and distributed data processing are hugely dominant in current research and development with conceptually new research results and software components emerging at the time of writing. In this article, we refer to several survey results, both for distributed data processing and for online machine learning.

Cross-References

- ▶ [Online Machine Learning Algorithms over Data Streams](#)
- ▶ [Recommender Systems Over Data Streams](#)
- ▶ [Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#)

Acknowledgements Support from the EU H2020 grant Streamline No 688191 and the “Big Data—Momentum” grant of the Hungarian Academy of Sciences.

References

- Abadi DJ, Carney D, Çetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. VLDB J 12(2):120–139
- Aggarwal CC (2013) A survey of stream clustering algorithms. In: Aggarwal CC, Reddy CK (eds) Data clustering: algorithms and applications. CRC Press, Boca Raton, p 231
- Aggarwal CC, Han J, Wang J, Yu PS (2003) A framework for clustering evolving data streams. In: Proceedings of the 29th international conference on very large data bases, vol 29. VLDB Endowment, pp 81–92
- Akida T, Balikov A, Bekiroğlu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) Millwheel: fault-tolerant stream processing at internet scale. Proc VLDB Endow 6(11):1033–1044
- Akida T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc VLDB Endow 8(12):1792–1803

- Arasu A, Babcock B, Babu S, Datar M, Ito K, Nishizawa I, Rosenstein J, Widom J (2003) Stream: the Stanford stream data manager (demonstration description). In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, pp 665–665
- Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, pp 1–16
- Babu S, Widom J (2001) Continuous queries over data streams. *ACM Sigmod Record* 30(3):109–120
- Bernhardt T, Vasseur A (2007) Esper: event stream processing and correlation. O'Reilly. ONJava, in <http://www.onjava.com/lpt/a/6955>
- Bifet A, Frank E (2010) Sentiment knowledge discovery in twitter streaming data. In: International conference on discovery science. Springer, pp 1–15
- Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) Moa: massive online analysis. *J Mach Learn Res* 11: 1601–1604
- Bifet A, Kirkby R, Pfahringer B (2011) Data stream mining: a practical approach. Technical Report, University of Waikato
- Bifet A, Zhang J, Fan W, He C, Zhang J, Qian J, Holmes G, Pfahringer B (2017) Extremely fast decision tree mining for evolving data streams. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1733–1742
- Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in apache flink®: consistent stateful distributed stream processing. *Proc VLDB Endow* 10(12):1718–1729
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink™: stream and batch processing in a single engine. *IEEE Data Eng Bull* 38:28–38
- Chambers C, Raniwala A, Perry F, Adams S, Henry RR, Bradshaw R, Weizenbaum N (2010) Flumejava: easy, efficient data-parallel pipelines. *ACM Sigplan Not* 45(6):363–375
- Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 785–794
- Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z (2015) MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*
- Cheng J, Ke Y, Ng W (2008) A survey on algorithms for mining frequent itemsets over data streams. *Knowl Inf Syst* 16(1):1–27
- Crankshaw D, Wang X, Zhou G, Franklin MJ, Gonzalez JE, Stoica I (2017) Clipper: a low-latency online prediction serving system. In: NSDI, pp 613–627
- De Francisci Morales G (2013) Samoa: a platform for mining big data streams. In: Proceedings of the 22nd international conference on world wide web. ACM, pp 777–778
- De Francisci Morales G, Bifet A, Khan L, Gama J, Fan W (2016) Iot big data stream mining. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 2119–2120
- Dean J, Corrado G, Monga R, Chen K, Devin M, Mao M, Senior A, Tucker P, Yang K, Le QV et al (2012) Large scale distributed deep networks. In: Advances in neural information processing systems. Neural Information Processing Systems Foundation, Inc., Lake Tahoe, pp 1223–1231
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 71–80
- Fan W, Bifet A (2013) Mining big data: current status, and forecast to the future. *ACM SIGKDD Explor News* 14(2):1–5
- Fitzpatrick B (2004) Distributed caching with memcached. *Linux J* 2004(124):5
- Fontenla-Romero Ó, Guijarro-Berdiñas B, Martínez-Rego D, Pérez-Sánchez B, Peteiro-Barral D (2013) Online machine learning. In: Igelnik B, Zurada JM (eds) Efficiency and scalability methods for computational intellect. IGI Global, Hershey, p 27
- Frigó E, Pálavics R, Kelen D, Benczúr AA, Kocsis L (2017) Online ranking prediction in non-stationary environments. In: Proceedings of the 1st workshop on temporal reasoning in recommender systems, co-located with 11th international conference on recommender systems
- Gaber M, Zaslavsky A, Krishnaswamy S (2007) A survey of classification methods in data streams. In: Aggarwal CC (ed) Data streams. Springer, New York, pp 39–59
- Gaber MM, Gama J, Krishnaswamy S, Gomes JB, Stahl F (2014) Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdiscip Rev Data Min Knowl Disc* 4(2):116–138
- Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. *Mach Learn* 90(3): 317–346
- Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv (CSUR)* 46(4):44
- Grossman RL, Hornick MF, Meyer G (2002) Data mining standards initiatives. *Commun ACM* 45(8):59–61
- Gualtieri M, Rowan Curran A, TaKeaways K, To MTBPP (2013) The forrester wave: Big data predictive analytics solutions, Q1 2013. Forrester research
- Henzinger MR, Raghavan P, Rajagopalan S (1998) Computing on data streams. *External Memory Algorithm* 50:107–118
- Ho Q, Cipar J, Cui H, Lee S, Kim JK, Gibbons PB, Gibson GA, Ganger G, Xing EP (2013) More effective distributed ML via a stale synchronous parallel parameter server. In: Advances in neural information processing systems, pp 1223–1231

- Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 97–106
- Jagerman R, Eickhoff C, de Rijke M (2017) Computing web-scale topic models using an asynchronous parameter server. In: Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval. ACM
- Jiang J, Cui B, Zhang C, Yu L (2017) Heterogeneity-aware distributed parameter servers. In: Proceedings of the 2017 ACM international conference on management of data. ACM, pp 463–478
- Jiang J, Yu L, Jiang J, Liu Y, Cui B (2017) Angel: a new large-scale machine learning system. *Natl Sci Rev* 5:216–236. nwx018
- Kavitha V, Punithavalli M (2010) Clustering time series data stream-a literature survey. arXiv preprint arXiv:1005.4270
- Kiran M, Murphy P, Monga I, Dugan J, Baveja SS (2015) Lambda architecture for cost-effective batch and speed big data processing. In: 2015 IEEE international conference on big data (Big Data). IEEE, pp 2785–2792
- Klinkenberg R, Joachims T (2000) Detecting concept drift with support vector machines. In: ICML, pp 487–494
- Lam W, Liu L, Prasad S, Rajaraman A, Vacheri Z, Doan A (2012) Muppet: mapreduce-style processing of fast data. *Proc VLDB Endow* 5(12):1814–1825
- Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY (2014a) Scaling distributed machine learning with the parameter server. In: 11th USENIX symposium on operating systems design and implementation
- Li M, Andersen DG, Smola AJ, Yu K (2014b) Communication efficient distributed machine learning with the parameter server. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds) Advances in neural information processing systems, vol 27. Curran Associates, Inc., New York, pp 19–27
- Li M, Zhou L, Yang Z, Li A, Xia F, Andersen DG, Smola A (2013) Parameter server for distributed machine learning. In: Big learning NIPS workshop, vol 6, p 2
- Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM (2012) Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proc VLDB Endow* 5(8):716–727
- Mahdiraji AR (2009) Clustering data stream: a survey of algorithms. *Int J Knowl based Intell Eng Sys* 13(2): 39–44
- Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning Publications Co., Shelter Island
- Morales GDF, Bifet A (2015) Samoa: scalable advanced massive online analysis. *J Mach Learn Res* 16(1): 149–153
- Muthukrishnan S et al (2005) Data streams: algorithms and applications. *Found Trends® Theor Comput Sci* 1(2):117–236
- Nair A, Srinivasan P, Blackwell S, Alcicek C, Fearon R, De Maria A, Panneerselvam V, Suleyman M, Beattie C, Petersen S et al (2015) Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296
- Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: Distributed stream computing platform. In: 2010 IEEE international conference on data mining workshops (ICDMW). IEEE, pp 170–177
- Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza stateful scalable stream processing at linkedin. *Proc VLDB Endow* 10(12):1634–1645
- Owen S, Anil R, Dunning T, Friedman E (2011) Mahout in action. Manning Publications, Greenwich
- Pálovics R, Kelen D, Benczúr AA (2017) Tutorial on open source online learning recommenders. In: Proceedings of the eleventh ACM conference on recommender systems. ACM, pp 400–401
- Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2009) Dataset shift in machine learning. The MIT Press, Cambridge
- Ranjan R (2014) Streaming big data processing in data-center clouds. *IEEE Cloud Comput* 1(1):78–83
- Schelter S, Satuluri V, Zadeh RB (2014) Factorbirda parameter server approach to distributed matrix factorization. In: NIPS 2014 workshop on distributed machine learning and matrix computations
- Shalev-Shwartz S et al (2012) Online learning and online convex optimization. *Found Trends® Mach Learn* 4(2):107–194
- Silva JA, Faria ER, Barros RC, Hruschka ER, de Carvalho AC, Gama J (2013) Data stream clustering: a survey. *ACM Comput Surv (CSUR)* 46(1):13
- Smola A, Narayananurthy S (2010) An architecture for parallel topic models. *Proc VLDB Endow* 3(1–2): 703–710
- Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J et al (2014) Storm @ Twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, pp 147–156
- Tsymbol A (2004) The problem of concept drift: definitions and related work. Technical Report 2, Computer Science Department, Trinity College Dublin
- White T (2010) Hadoop: the definitive guide. Yahoo Press, Cambridge
- Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. *Mach learn* 23(1):69–101
- Xing EP, Ho Q, Dai W, Kim JK, Wei J, Lee S, Zheng X, Xie P, Kumar A, Yu Y (2015) Petuum: a new platform for distributed machine learning on big data. *IEEE Trans Big Data* 1(2):49–67
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. *HotCloud* 10(10–10):95
- Zhou J, Cui Q, Li X, Zhao P, Qu S, Huang J (2017) PSMART: parameter server based multiple additive regression trees system. In: Proceedings of the 26th international conference on world wide web companion, pp 879–880. International World Wide Web Conferences Steering Committee

- Zhu Y, Shasha D (2002) Statstream: statistical monitoring of thousands of data streams in real time. In: Proceedings of the 28th international conference on very large data bases. VLDB Endowment, pp 358–369
- Žliobaite I, Bifet A, Gaber M, Gabrys B, Gama J, Minku L, Musial K (2012) Next challenges for adaptive learning systems. ACM SIGKDD Explor News 14(1): 48–55

Online Machine Learning on Streams

- ▶ Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning
- ▶ Recommender Systems over Data Streams
- ▶ Online Machine Learning Algorithms over Data Streams
- ▶ Online Machine Learning in Big Data Streams: Overview

Online Process Discovery

- ▶ Streaming Process Discovery and Conformance Checking

Online Process Mining

- ▶ Streaming Process Discovery and Conformance Checking

Ontologies for Big Data

Eva Blomqvist
Linköping University, Linköping, Sweden

Synonyms

Classically, in AI, ontologies were considered a form of knowledge representation and frequently

denoted knowledge models. More recent near-synonyms include the term *vocabulary*, which is usually used for expressive ontologies on the Web that are used as vocabularies for online datasets. Even more recently the term *knowledge graph* has gained increased popularity. It can be viewed as a more general term, since a knowledge graph does not necessarily have to be an ontology; however, most ontologies could be considered to be knowledge graphs.

Overview

This chapter introduces the concept of ontology, in the context of computer science and in particular Big Data. First, some terminology is introduced and the concept is defined. Further, some background and history of ontologies in computer science is presented. Next, common representation formats and standards are introduced, followed by a discussion on methods for constructing, finding, and reusing ontologies. The chapter is concluded by discussing the contribution of ontologies to Big Data solutions, as well as current research challenges for ontologies.

Definitions

An ontology in the computer science field can be defined as “a set of representational primitives with which to model a domain of knowledge or discourse” (Gruber 2009), where such primitives include concepts (or classes) and relations (attributes, properties, or other relations between entities). It is also commonly described as a formal specification of a shared conceptualization. Where a conceptualization is an abstract representation of some actual phenomenon or state of affairs, e.g., the concept of cat is usually understood by everyone, that conceptualization is a different thing from the actual cats that exist in the world that are probably not even aware of being grouped under that abstract concept. Formal means that it should be represented in some machine interpretable language with well-defined semantics, most commonly a logic-based

language that facilitate some logical reasoning over the models. Specification refers to that it contains definitions, perhaps in the form of logical axioms, of the concepts it is comprised of. Finally, shared means that it should usually not represent only the viewpoint of a single individual, but rather be shared among users and/or system instances within some domain and/or for some specific task.

Background and Origin

The term ontology originates in ancient Greek, where *on* (genitive *ontos*) is the verb “being” and *logia* (originally derived from *logos*) is the term for “science” or “study” ([Nationalencyklopedien](#)). Ontology as a term could thereby be interpreted as “the study of being,” and since the seventeenth century, the term *ontology* has been used to denote the field of metaphysics devoted to the study of the nature of being (Simons [2009](#)). In more recent times, the term ontology has been adopted by the computer science field and now denotes a formal structure that explicitly specifies the concepts existing within some domain or related to a specific application. In contrast to the original meaning, ontologies are today often used in a less prescriptive way, i.e., ontologies in computer science do not primarily deal with the inner “true” nature of reality; instead they are a means of *describing* a domain specifically tailored for a task at hand, i.e., to be used as a knowledge representation within some computer system.

Even in the field of computer science, there are however numerous definitions of the term ontology in the literature, still today existing in parallel. Apart from the one referenced above, in the definitions section, among the most commonly used definitions we find:

- “An ontology is an explicit specification of a conceptualization.”(Gruber [1993](#))
- “An ontology is a formal, explicit specification of a shared conceptualization.” (Studer et al. [1998](#))
- “An ontology is a logical theory accounting for the intended meaning of a formal vocab-

ulary, i.e., its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment by approximating these intended models.” (Guarino [1998](#))

An ontology corresponds to a specific *domain* of knowledge, it has a scope and is thereby focused on describing a certain domain (although sometimes the domain can be very broad, and sometimes as narrow as a single software application). The domain can be an industry domain, an enterprise, a research field or any other imaginable restricted set of knowledge, whether abstract, concrete or even imagined or envisioned. Additionally the ontology is constructed with a certain *usage* in mind, this usage target sets additional restrictions on the content and structure of the ontology. An ontology may, for example, be used to classify instances, check consistency, or answer queries, but on the other hand, there may be different kinds of uses, like communicating and sharing precise definitions of terms within a community of human practitioners. The nature or the intended use sets requirements on the content and structure of the ontology.

Formally, an ontology that is represented in a logical language is simply a set of logical axioms expressed in that language. However, in practice, it is often useful to think about an ontology as consisting of a set of basic building blocks. The most basic building blocks of an ontology are its concepts, also commonly denoted *classes*. There is no consensus around the definition of what an ontology concept really is, but generally a concept can be seen as having three parts (according to Cimiano [2006](#)):

- an intentional definition,
- an extension,
- a set of lexical realizations.

Where the intentional definition is a definition or description of the meaning of the concept, sometimes this is simply represented as a natural language description of the concept but could

also be the logical axioms involving the concept. The extension contains the instances of the concept, the set of members of this category of entities, and the set of objects that exemplify the notion represented by the concept. The set of lexical realizations is a set of terms that may be used to represent the concept, e.g., both the terms “packet” and “parcel” may be used to represent the conceptual notion of a package in some ontology. Sometimes a lexical realization is also called a label of the concept, the concept may thereby have a set of alternative labels, and the labels may very well be from different languages.

Another basic element is the relation. A relation is some kind of association that exists between two or more instances, although in most cases ontology representation languages restrict this to modelling only binary relations. In general a relation can be viewed as a tuple of instances, but the properties of the relation should be described in the ontology. One such description can be some lexical realizations of the relation, i.e., relation labels, and possibly a set of axioms determining the semantic interpretation of the relation. For example, the special type of relation usually denoted subsumption, taxonomical relation, or “subclass of” has several lexical realizations, such as “subsumption,” “subclass-of,” and “is_a.” This type of relation additionally comes with some specific semantics, for example, the relation is commonly defined to be transitive, and its domain and range can usually be any concept in an ontology. For specific types of relations, such as the “subclass-of,” that are predefined in many ontology representation languages, these semantics are inherent in the semantics of the logical language, but for other relations, they need to be specified explicitly in the ontology, if needed for some reasoning task.

Ontologies can also be classified from the point of view of their level or generality, as described by Guarino (1998). Top-level ontologies describe general concepts like space and time, which are most often independent of application domains and specific problems. These ontologies can be shared by large communities. Examples of top-level ontologies include DOLCE (<http://www.loa.istc.cnr.it/old/DOLCE.html>)

and its more popular ultra-light version DUL (http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite), SUMO (<http://www.adampease.org/OP/>), BFO (<http://ontology.buffalo.edu/bfo/>), and CYC (<http://www.cyc.com/>). Domain ontologies describe a vocabulary of a specific domain and are sometimes specializations of top-level ontologies. Task ontologies in turn describe the vocabulary of a generic task or activity and may also be specializations of top-level ontologies. Finally, an application ontology is a specialization of a domain ontology adapted for a specific application or task.

Another classical distinction was the separation between the ontology and its knowledge base or in logical terms between the TBox and the ABox. While earlier this was a clear distinction, today, and in particular on the Web, this distinction has been blurred. Today ontology and knowledge base is used almost interchangeably, and an ontology can many times be comprised of both concepts and instances, i.e., both TBox and ABox. Instead the distinction is made between ontologies and datasets, where ontologies are seen as the schemas that describe the content of the dataset, but where this schema can also contain sets of instances. An example of such a case would be an ontology that models a set of codes used in the data as instances of an ontology concept. In this case, the codes and the description of their intended meaning is seen as part of the ontology, regardless if they are modelled as individuals or classes. The codes are then referenced and used in the dataset.

Ontology Representation and Reasoning

Many different logical formalisms exist for representing and reasoning with ontologies, each one having its own benefits and drawbacks. The choice of representation should be made based on the requirements of the ontology, but additionally things like standardization and tool availability may influence such a decision. In the early days of ontology engineering, two different traditions

co-existed; while some people preferred frame-style logics that are more closely related to traditional information modelling, others preferred to use description logics (DL) related formalisms. However, with the emergence of the Semantic Web community in the early 1900s, and the W3C standards emerging from that, the predominant formalism is now DLs, and most commonly the W3C recommendation OWL (Web Ontology Language) (OWL Working Group 2012). Nevertheless, other languages (and OWL extensions) for specific purposes, such as temporal reasoning, fuzzy, or probabilistic reasoning, have also emerged but are much less common in practical applications.

The Resource Description Framework (RDF) (RDF Working Group 2014) is another W3C recommendation, for representing data and metadata about resources on the Web, or, for instance, to represent the data associated with an OWL ontology. The RDF model is based on triples, consisting of a subject, a predicate, and an object. This is how we can talk about resources in RDF, i.e., state facts, through expressing triples connecting the resource to other resources or literals, i.e., forming an RDF graph. URIs are used to identify resources; hence, this is a Web-targeted formalism, but which can equally well be used in other settings. RDF has several serializations, e.g., both an XML syntax (Gandon et al. 2014) and several more human readable formats, such as turtle (Beckett et al. 2014), into which also OWL ontologies can be serialized.

RDF Schema (RDFS) (Brickley et al. 2014) is an extension of RDF for defining schemas to be used by RDF data. With RDFS one can build a simple ontology, for example, defining named classes (concepts) using `rdfs:Class` and a taxonomy of classes using `rdfs:subClassof`. OWL builds on RDF(S) in the sense that it extends the RDFS syntax with further features but additionally restricts RDFS to make automated reasoning feasible. Both OWL and RDFS can syntactically be represented as an RDF graph.

For performing automated reasoning over OWL ontologies, a number of inference engines exist, where one list is maintained by the

W3C (2016). Many triple stores, i.e., database management systems tailored for storing and serving RDF data, also include inference engines supporting subsets of the OWL language. Due to its DL roots and its focus on the open Web, OWL has however a number of specific features that have been criticized throughout the years. This includes, for instance, the open-world assumption, which makes the OWL semantics unintuitive to many people more familiar with traditional relational databases or information modelling and which makes it complex and sometimes unfeasible to perform constraint checking over data. Therefore, a W3C working group recently published a new recommendation, the RDF Shapes Constraints Language (SHACL) (Knublauch and Kontokostas 2017). The language is intended for describing RDF data, as well as checking constraints over those data, i.e., an area that is poorly covered by the OWL language. Although it can be debated if SHACL should be considered an ontology language or not, it is nevertheless certainly an important complement to OWL when describing and reasoning over data.

Ontology Engineering and Reuse

Methodologies for creating ontologies have existed as long as ontologies have been used in the context of AI. Traditionally, such methodologies were based on knowledge engineers performing some kind of knowledge acquisition directly from domain experts and then formalizing this knowledge into an ontology. Most early methods were “manual” in the sense that they did not rely on any tool support or automated processes. In the 1990s and 2000s, there were many attempts at drawing from software engineering in order to somewhat formalize and structure such methodologies. Examples of commonly referenced ontology engineering methodologies include METHONTOLOGY (Fernández et al. 1997), ontology development 101 (Noy and McGuinness 2001), the method for developing enterprise ontologies by Grüniger and Fox (1995) (also the first to introduce the notion of

competency questions as ontology requirements), and the NeOn family of methods (Suárez-Figueroa et al. 2012). More recent methodologies commonly focus on an agile process, such as eXtreme Design (Blomqvist et al. 2016), or on the iterative evolution of ontologies, such as DILIGENT (Pinto et al. 2009).

For building ontologies, several tools have been proposed during the past decades (see also the list by W3C 2016); however, today two tool suites could be viewed as the most popular and widely used; the Protégé tool suite (including both the desktop version and the online collaborative WebProtégé environment) and TopBraid Composer as part of TopQuadrant's suite of tools. While the latter is commercial and provides many features and components also related to managing and transforming RDF data, the former is open source and targeted entirely at OWL ontologies, but with a large user community and many interesting tool plugins.

Another popular approach is, rather than building an ontology by hand, to try to derive ontologies from other resources. There is a large body of work on deriving ontologies from other structured sources, such as XML schemas, relational databases, tables and CSV-files, etc. Other approaches try to derive ontologies automatically from unstructured sources, such as natural language texts, so-called ontology learning. Although many approaches have been proposed, there is still a gap in terms of building a suitable ontology for a set of requirements out of such resources; hence, such learned ontologies are only suitable for certain tasks and will usually produce quite shallow and logically inexpressive ontologies, i.e., rather lists of terms or simple taxonomies than expressive OWL models.

There are many best practices available when constructing and publishing ontologies. Some have emerged in the form of Ontology Design Patterns (Hitzler et al. 2016), while others are included in documents describing how to publish and manage your Web data (Dodds and Davis 2012) and W3C guideline documents (Kendall et al. 2008) or as part of the language recommendations themselves.

When reusing existing large ontologies, a common task is also to align two or more ontologies. Ontology matching and alignment (Euzenat and Shvaiko 2013) is the task of finding correspondences, e.g., overlap or relations, between the ontologies. A number of automated approaches have been proposed for this task, but in the end, it is usually recognized that a human expert needs to be involved to validate the produced alignment, check for problems, and repair issues found.

Commonly Used Vocabularies

As mentioned earlier, for a very specific use case, it is rare to find one ontology or a set of ontologies, that completely cover all the requirements. Nevertheless, there are some standard ontologies that are frequently reused and extended, which are worth mentioning. The W3C has in recent years chosen to recommend a set of vocabularies for data on the Web. This set includes:

- The RDF data cube vocabulary (<https://www.w3.org/TR/vocab-data-cube/>) – intended for representing multi-dimensional data on the web using RDF, such as large statistical datasets.
- The Prov-O (<https://www.w3.org/TR/prov-o/>) – intended for representing provenance about data on the Web.
- The data catalogue vocabulary DCAT (<https://www.w3.org/TR/vocab-dcat/>) – intended for representing metadata about a dataset or catalogue.
- The organization ontology (<https://www.w3.org/TR/vocab-org/>) – intended as a common vocabulary for publishing data about organizations.
- The time ontology (<https://www.w3.org/TR/owl-time/>) – intended for expressing temporal concepts in a standardized manner.
- The Semantic Sensor Network ontology (SSN) (<https://www.w3.org/TR/vocab-ssn/>) – intended for representing information about sensor data and IoT systems, such as data about observations and the sensors that made the observations.

In addition, the standardization organization OGC has also published a standard for representing and querying geographical data on the Web, GeoSPARQL (<http://www.opengeospatial.org/standards/geospqrql>), which includes a vocabulary for representing geographical features and their attributes.

Apart from standard vocabularies, there are also numerous well-known ontologies, such as the top-level (upper) ontologies mentioned earlier, as well as specific reference ontologies in various domains, such as Snomed-CT (<https://www.snomed.org/snomed-ct>) and the Gene ontology (<http://www.geneontology.org/>) in the medical field. A collection of such vocabularies, with references to their usage, can be found at the Linked Open Vocabularies site (<http://lov.okfn.org/dataset/lov/>). However, there are also domain specific catalogues available, such as BioPortal for the biomedical domain (<https://bioportal.bioontology.org/>).

Ontology Usage for Big Data

Ontologies can be used for many tasks. While, in the earlier days of AI, ontologies were often used in isolation, providing the logical machinery for some reasoning task that was central to the application, such as an expert system, today most ontologies are used in combination with large datasets and potentially other classes of methods, such as natural language processing (NLP) or machine learning (ML) approaches. A few of the tasks for which ontologies can be applied include:

- Data integration – acting as a unifying model for representing and linking diverse datasets.
- Data access – acting as vocabularies for understanding and querying datasets, regardless whether those datasets are integrated and stored in one place or queries are simply translated and distributed to the original data sources, as well as allowing for graph traversal to explore and analyze the datasets.
- Semantic search – providing the opportunity to refine or expand a keyword query in a structured manner or translating a search query into a structured query over a dataset.

- Data analysis, cleaning, and constraint checking – providing logical axioms for performing data analysis, allowing analytical queries, or checking quality or other constraints over data.
- Reasoning for deriving new information – providing logical axioms for deriving relevant conclusions from the available data.
- Semantic similarity computations – allowing for computing the similarity between individual data (or ontology) elements, as well as sets of elements representing, for instance, the attributes of an entity.
- Ontology-based information extraction – supporting the extraction of new structured information from natural language text, for populating its knowledge base or another dataset.
- Integration with ML approaches – ontologies can, for instance, be used as a structure of the potential input features of a ML method or provide the output structure, e.g., classes, to which some input is automatically classified.

When working with diverse data sources on the Web, there are two main directions of approaches for integrating data, i.e., usually depending on whether data will be transformed into a common format, loaded and stored in a central storage, or if data is to still be kept in distributed storage facilities and only accessed through a common interface. In the former case, a common pattern is to create (or reuse) a shared unifying ontology (or several ontologies), into which all the data to be loaded is mapped and then transformed into a common format, such as RDF. Data can then be stored and retrieved from a single storage and queried by referring to the unifying ontology (or ontologies). However, when considering big data, this will rarely be a feasible approach, i.e., to load all the data into the same storage (even if a distributed one). Another approach is then to instead simply map relevant parts of the schemas or data models of the original data sources into the common unifying ontology (or ontologies) and to use these

mappings for generating queries to the original data sources, based on any query or reasoning task expressed over the unifying ontology (or ontologies). Approaches in this direction can today be found under the label ontology-based data access (OBDA). Several mapping languages exist, for instance, between relational models and ontologies, such as the R2RML (a W3C recommendation) (Das et al. 2012). However, this is still an area of active research, especially when it comes to efficiency of query distribution and execution over diverse sources.

Apart from this, ontologies are already used in large-scale settings on the Web under the term *knowledge graphs*. The most prominent example of this, and the application that popularized the term knowledge graph, is of course the Google knowledge graph (Singhal 2012). The Google knowledge graph is an ontology coupled with a large dataset, used to interpret queries and enrich returned results in the Google search engine. The Google knowledge graph contains a general-purpose ontology, intended to cover all possible domains of interest, but clearly focused on common entity categories, such as persons, organizations, places, events, etc. When such a category of interest is discovered from a search query, Google is able to use the knowledge graph to “understand” what kind of answers the user is looking for, and reply in a more precise manner, rather than a simple list of pages with snippets. For instance, when searching for a famous person on Google, the result usually contains an additional small table with data about that person, next to the usual hit list with pages. This is generated based on the knowledge graph. Following the example of Google, recently many other large companies have tried a similar path, creating their own knowledge graphs, such as Facebook’s experiments with their Graph Search (Olanoff et al. 2013), Microsoft’s entity engine (Lardinois 2014), and lately developing enterprise knowledge graphs at Siemens (Fishkin 2018). Even before Google announced their knowledge graph, IBM had already used similar approaches, combined with advanced NLP techniques, in their success with IBM Watson (<https://www.ibm.com/watson/index.html>), the

Jeopardy-winning general-purpose question answering system.

Research Challenges

Current research challenges, regarding ontologies, include finding the balance between carefully engineered knowledge representations, such as an ontology, and other kinds of models, for instance, learned through machine learning methods. While ontologies are transparent and can easily provide explanations for a derived conclusion, it is often not feasible to create a detailed model of the complete domain and task by hand, and it may not even be the case that the knowledge exists at design time. Rather, some parts of it may need to be discovered from incoming data or user interactions and modified as the system runs. This is nowadays recognized as one of the main research integration challenges in AI, i.e., to integrate the knowledge representation and reasoning approach with state-of-the-art machine learning methods in order to take advantage of both.

A related problem is the challenge of coverage and fit of ontologies that exist today. Despite many ontologies being published and shared today, it is still rare to find an existing ontology that perfectly fits your needs. Usually it needs to be combined with other ontologies or extended with specifics of your use case. Hence, despite the fact that many ontologies exist, one has to expect that a certain amount of extension is usually needed in order to use an ontology or that parts of several ontologies are needed to annotate a dataset. Therefore, there is still a lot of ongoing research regarding the tasks of finding, evaluating, and reusing ontologies, as well as aligning several ontologies into an ontology network. Additionally, the challenge of updating ontologies, e.g., through ontology evolution, is also an ongoing research topic.

In particular when using ontologies for big data applications, the issue of scale, i.e., both volume and velocity, is an important aspect. Although some ontologies are potentially too expressive to be used for large-scale inferencing,

there are both subsets of the OWL language where efficient reasoning methods exist and methods for performing incremental reasoning or stream reasoning using windows over data, which can make OWL ontologies applicable in large-scale settings. However, this challenge is still a topic of research as well.

Nevertheless, regardless of the actual representation formalisms, ontologies in the form of knowledge graphs have already proven their usefulness and applicability in big data settings, such as operating a search engine over the entire Web in the case of Google.

Cross-References

- ▶ [Automated Reasoning](#)
- ▶ [Reasoning at Scale](#)

References

- Beckett D, Berners-Lee T, Prud'hommeaux E, Carothers G (2014) RDF 1.1 turtle. <https://www.w3.org/TR/2014/REC-turtle-20140225/>
- Blomqvist E, Hammar K, Presutti V (2016) Engineering ontologies with patterns – the extreme design methodology. In: Hitzler P, Gangemi A, Janowicz K, Krisnadhi A, Presutti V (eds) *Ontology engineering with ontology design patterns, studies on the semantic web*, vol 25. IOS Press, Amsterdam
- Brickley D, Guha R, McBride B (2014) RDF schema 1.1. <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>
- Cimiano P (2006) *Ontology learning and population from text – algorithms, evaluation and applications*. Springer, New York
- Das S, Sundara S, Cyganiak R (2012) R2RML: RDB to RDF mapping language. <https://www.w3.org/TR/r2rml/>
- Dodds L, Davis I (2012) Linked data patterns – a pattern catalogue for modelling, publishing, and consuming linked data. <http://patterns.dataincubator.org>
- Euzenat J, Shvaiko P (2013) *Ontology matching*. Springer, Berlin
- Fernández M, Gómez-Pérez A, Juristo N (1997) Methontology: from ontological art towards ontological engineering. In: Proceedings of the AAAI97 spring symposium series on ontological engineering
- Fishkin A (2018) Industrial knowledge graph at siemens – powered by metaphactory and Amazon Neptune. Presentation at CERN openlab technical workshop, Geneva. https://indico.cern.ch/event/669648/contributions/2838194/attachments/1581790/2499984/CERN_Open_Lab_Technical_Workshop_-_SIEMENS_AG_-_FISHKIN_-_11-01-2018.pdf
- Gandon F, Schreiber G, Beckett D (2014) RDF 1.1 XML syntax. <https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/>
- Gruber T (1993) A translation approach to portable ontology specifications. *Knowl Acquis* 5(2):199–220
- Gruber T (2009) Ontology. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, New York
- Grüninger M, Fox MS (1995) Methodology for the design and evaluation of ontologies. In: Workshop on basic ontological issues in knowledge sharing, IJCAI-95, Montreal
- Guarino N (1998) Formal ontology and information systems. In: *Formal ontology in information systems. Proceedings of FOIS'98*, Trento, 6–8 June 1998. IOS Press, pp 3–15
- Hitzler P, Gangemi A, Janowicz K, Krisnadhi A, Presutti V (eds) (2016) *Ontology engineering with ontology design patterns: foundations and applications. Studies on the semantic web*, vol 25. IOS Press, Amsterdam
- Kendall E, Novacek V, Baker T, Miles A (2008) Principles of good practice for managing RDF vocabularies and OWL ontologies. <https://www.w3.org/2006/07/SWD/Vocab/principles>
- Knublauch H, Kontokostas D (2017) Shapes constraint language (SHACL). <https://www.w3.org/TR/shacl/>
- Lardinois F (2014) Microsoft has big plans for bing's entity engine. <https://techcrunch.com/2014/03/30/microsoft-has-big-plans-for-bings-entity-engine/>
- Nationalencyklopedin. *Ontologi*. In: Nationalencyklopedin. Nationalencyklopedin. <http://www.ne.se/uppslagsverk/encyklopedi/lng/ontologi>. Accessed 30 Jan 2018
- Noy NF, McGuinness DL (2001) *Ontology development 101: a guide to creating your first ontology*. Stanford knowledge systems laboratory technical report and Stanford Medical Informatics technical report KSL-01-05 and SMI-2001-0880, Stanford Knowledge Systems Laboratory
- Olanoff D, Constine J, Taylor C, Lunden I (2013) Facebook announces its third pillar “graph search” that gives you answers, not links like Google. <https://techcrunch.com/2013/01/15/facebook-announces-its-third-pillar-graph-search/>
- OWL Working Group (2012) Web ontology language (OWL). <https://www.w3.org/OWL/>
- Pinto HS, Tempich C, Staab S (2009) Ontology engineering and evolution in a distributed world using diligent. In: Staab S, Studer R (eds) *Handbook on ontologies*. Springer, New York
- RDF Working Group (2014) Resource description framework (RDF). <https://www.w3.org/RDF/>
- Simons PM (2009) Ontology. In: Encyclopædia Britannica. Encyclopædia Britannica. <https://www.britannica.com/topic/ontology-metaphysics>
- Singhal A (2012) Introducing the knowledge graph: things, not strings. <https://googleblog.blogspot.se/2012/05/introducing-knowledge-graph-things-not.html>

- Studer R, Benjamins VR, Fensel D (1998) Knowledge engineering: principles and methods. *Data Knowl Eng* 25:161–197
- Suárez-Figueroa M, Gómez-Pérez A, Motta E, Gangemi A (eds) (2012) Ontology engineering in a networked world. Springer, Berlin/Heidelberg
- W3C (2016) OWL/implementations. <https://www.w3.org/2001/sw/wiki/OWL/Implementations>

Ontology-Based Data Access

- ▶ Integration-Oriented Ontology

Operational Analytics

- ▶ Hybrid OLTP and OLAP

Optimizing Geo-Distributed Streaming Analytics

Abhishek Chandra¹, Benjamin Heintz¹, and Ramesh Sitaraman²

¹University of Minnesota, Minneapolis, MN, USA

²University of Massachusetts, Amherst, MA, USA

Overview

Data analytics is undergoing a revolution: the volume and velocity of data sources are increasing rapidly. Across a number of application domains from web, social, and energy analytics to scientific computing, large quantities of data are generated continuously in the form of posts, tweets, logs, sensor readings, and more. A modern analytics service must provide real-time analysis of these data streams to extract meaningful and timely information. As a result, there has been a growing interest in streaming analytics with recent development of several distributed analytics platforms (Apache Storm 2015; Boykin et al. 2014; Zaharia et al. 2013).

In many streaming analytics domains, inputs originate from diverse sources including users, devices, and sensors located around the globe. As a result, the distributed infrastructure of a typical analytics service (e.g., Google Analytics, Akamai Media Analytics, etc.) has a hub-and-spoke model. Data sources send streams of data to nearby “edge” servers. These geographically distributed edge servers send data to a central location that can process the data further, store summaries, and present those summaries in visual form to users of the analytics service. While the central hub is typically located in a well-provisioned data center, resources may be limited at the edge locations. In particular, the available WAN bandwidth between the edges and the center is limited.

A traditional approach to analytics processing is the *centralized model* where no processing is performed at the edges and all the data is sent to a dedicated centralized location. This approach is generally suboptimal, because it strains the scarce WAN bandwidth available between the edges and the center, leading to delayed results. Further, it fails to make use of the available compute and storage resources at the edge. An alternative is a *decentralized approach* (Rabkin et al. 2014) that utilizes the edge for much of the processing in order to minimize WAN traffic. However, neither approach is optimal for modern analytics requirements. Instead, analytics processing must utilize *both* edge and central resources in a carefully coordinated manner in order to achieve the stringent requirements of an analytics service in terms of network traffic, user-perceived delay, as well as accuracy of results.

A crucial question for a geo-distributed analytics system is how best to utilize resources at both the edges and the center in order to deliver timely results. In particular, optimizing geo-distributed streaming analytics requires an analytics system to determine how much computation to perform at the edges and how much to leave for the center (i.e., *where* to compute), as well as *when* to send partial results from edges to the center. These questions must be addressed in the context of the trade-off between multiple metrics: cost (e.g.,

WAN traffic), timeliness (e.g., latency), and data quality (e.g., result accuracy).

Key Research Findings

Recent work (Heintz et al. 2015, 2016a, 2017) has addressed the problem of optimizing geo-distributed streaming analytics by analyzing these trade-offs in the context of *windowed grouped aggregation*, an important primitive in any stream analytics system. This work systematically analyzes these trade-offs both in the context of exact computation (where all data must be completely processed) and approximate computation (where some error can be tolerated).

Abstractions for grouped aggregation are provided in most data analytics frameworks, for example, as the Reduce operation in MapReduce or Group By in SQL and LINQ. A useful variant in stream computing is *windowed* grouped aggregation, where each group is further broken down into finite time windows before being summarized. Windowed grouped aggregation is one of the most frequently used primitives in an analytics service and underlies queries that aggregate a metric of interest over a time window. Example queries that utilize windowed grouped aggregation include computing content popularity for a web analytics application or the average network load distribution for a network monitoring application.

Exact Computation

Recent work (Heintz et al. 2015, 2017) focuses on designing algorithms for performing windowed grouped aggregation in order to optimize the two key metrics of any geo-distributed streaming analytics service: *WAN traffic* and *staleness* (the delay in getting the result for a time window). The aggregation algorithm runs on each edge, aggregating the records in the input stream and sending (partial) aggregates to the center over the WAN. The scheduling problem is to determine when to send which aggregates to the center.

This work examines baseline approaches such as pure streaming, a centralized approach where all data is immediately sent from edge to center without any edge processing, and pure batching, a decentralized approach where all data during a time window is aggregated at the edge, with only the aggregated results being sent to the center at the end of the window. It shows that such approaches do not jointly optimize traffic and staleness and, hence, are suboptimal. It presents a family of *optimal offline algorithms* that *jointly minimize* both staleness and traffic. One offline optimal algorithm is the *eager optimal algorithm* that eagerly flushes updates for each distinct key immediately after the final arrival to that key within a window. Another offline optimal algorithm is the *lazy optimal algorithm* which schedules its updates to start at the last possible time that would still enable it to flush all its updates with an optimal staleness. There is a family of optimal algorithms whose schedules consist of update times that lie between those for the eager and lazy algorithm for each key.

Using these offline optimal algorithms as a foundation, practical online aggregation algorithms are developed that emulate the offline optimal algorithms. The key insight here is that windowed grouped aggregation can be modeled as a *caching problem* where the cache size varies over time. This insight allows the decomposition of the scheduling problem into two subproblems: determining the (dynamic) *cache size* and defining a *cache eviction policy*. While the first subproblem can be solved by using insights gained from the optimal offline algorithms, the second subproblem lends itself to using the vast prior work on cache replacement policies (Podlipnig et al. 2003). To overcome potential problems with pure emulation of either the eager or the lazy offline optimal algorithms in practice, a *hybrid algorithm* is proposed that computes cache size as a linear combination of eager and lazy cache sizes. Concretely, a hybrid algorithm with a *laziness parameter* α – denoted by $\text{hybrid}(\alpha)$ – estimates the cache size $c(t)$ at time t as:

$$c(t) = \alpha \cdot c_l(t) + (1 - \alpha) \cdot c_e(t),$$

where $c_l(t)$ and $c_e(t)$ are the lazy and eager cache size estimates, respectively. By selecting an appropriate value for α , the hybrid algorithm can achieve the desired trade-off between traffic and staleness.

The practicality of these algorithms is demonstrated through an implementation in Apache Storm (2015), deployed on the PlanetLab (2015) testbed. The experiments are driven by workloads derived from anonymized traces of a popular web analytics service offered by Akamai (Nygren et al. 2010), a large content delivery network. The results of the experiments show that the proposed online hybrid aggregation algorithms simultaneously achieve traffic close to optimal while reducing staleness significantly compared to baseline algorithms such as batching. Further, these algorithms are robust to a variety of system configurations (number of edges), stream arrival rates, and query types.

Approximate Computation

In a geo-distributed setting, due to constrained WAN bandwidth, it is not always feasible to produce exact results in a timely manner. In such cases, applications must either sacrifice timeliness by allowing delayed – i.e., late – results or sacrifice accuracy by tolerating some error in the results. This is a fundamental trade-off: it is not always feasible to compute exact results with bounded staleness (Rabkin et al. 2014). Further, many real-world applications can tolerate some staleness or inaccuracy in their final results.

Recent work (Heintz et al. 2016a) has studied the *staleness-error trade-off* for windowed grouped aggregation in geo-distributed streaming analytics, recognizing that applications have diverse requirements: some may tolerate higher staleness in order to achieve lower error and vice versa. As in the exact computation scenario, the aggregation algorithm runs on each edge, partially aggregating the input data and sending results to the center over the WAN. In this case, the scheduling problem is to determine when, and if, aggregates need to be sent to the center. Aggregation algorithms are devised to solve *two complementary problems*: minimize staleness un-

der an error constraint and minimize error under a staleness constraint.

This work first designs theoretically optimal offline algorithms to solve each of these problems and then develops practical online algorithms based on the intuition derived from the offline optimal algorithms. The optimal offline algorithms allow minimizing staleness (resp., error) under an error (resp., staleness) constraint. Intuitively, the error-bound problem is solved by transmitting only the aggregates that are essential to meet the error constraint. For the staleness-bound problem, error is minimized by prioritizing keys for transmission, based on their potential error (if not sent), while fully utilizing the available network bandwidth.

Using these offline algorithms as references, practical online algorithms are designed to efficiently trade off staleness and error. These practical algorithms are based on the key insight of representing grouped aggregation at the edge as a *two-part cache*. This formulation generalizes the caching-based framework for exact windowed grouped aggregation (Heintz et al. 2015) by introducing cache *partitioning* and cache *eviction* policies to identify which partial results must be sent and which ones can be discarded.

The practicality and efficacy of these algorithms is demonstrated through both trace-driven simulations and implementation in Apache Storm (2015), deployed on a PlanetLab (Peterson et al. 2003) testbed. Using workloads derived from traces of a popular web analytics service offered by Akamai (Nygren et al. 2010), experiments show that the proposed algorithms reduce staleness and error significantly compared to a practical aggregation algorithm that effectively combines batching with streaming using random sampling. The proposed techniques apply across a diverse set of aggregates, from distributive and algebraic aggregates (Gray et al. 1997) such as Sum and Max to sketch-based approximation of holistic aggregates such as unique count (via sketch data structures such as HyperLogLog Flajolet et al. 2007). Further, they can handle diverse network bandwidth and workload conditions and are adaptive to variable network conditions.

Related Work

Numerous stream computing systems (Akidau et al. 2013; Chandrasekaran et al. 2003; Kulkarni et al. 2015; Qian et al. 2013; Zaharia et al. 2013; Apache Flink 2016; Chen et al. 2016) have been proposed in recent years. The Google Dataflow model (Akidau et al. 2015) and the Apache Beam project (2016) have presented a unified abstraction for computing over both bounded and unbounded datasets. These systems provide many useful ideas for new analytics systems to build upon, but they are primarily designed for tightly coupled computing environments such as clusters and data centers and do not fully consider the challenges in a geo-distributed environment.

Wide-area computing has received increased research attention in recent years, due in part to the widening gap between data processing and communication costs. Much of this attention has been paid to batch computing (Pu et al. 2015; Vulimiri et al. 2015a,b; Heintz et al. 2016b). Relatively little work on streaming computation has focused on wide-area deployments or associated questions such as where to place computation. Pietzuch et al. (2006) optimize operator placement in geo-distributed settings to balance between system-level bandwidth usage and latency. Hwang et al. (2008) rely on replication across the wide area in order to achieve fault tolerance and reduce straggler effects.

JetStream (Rabkin et al. 2014) considers wide-area streaming computation and addresses the tension between timeliness and accuracy, focusing at a higher level on the appropriate abstractions for navigating this trade-off. Meanwhile BlinkDB (Agarwal et al. 2013) provides mechanisms to trade accuracy and response time, though it does not focus on processing streaming data. Das et al. (2014) consider trade-offs between throughput and latency in Spark Streaming, but they focus on exact computation and consider only a uniform batching interval for the entire stream.

Aggregation is a key operator in analytics, and grouped aggregation is supported by many

data-parallel programming models (Boykin et al. 2014; Gray et al. 1997; Yu et al. 2009). Larson et al. (2002) explore the benefits of performing partial aggregation prior to a join operation. While they also recognize similarities to caching, they consider only a simple fixed-size cache. In sensor networks, aggregation is often performed over a hierarchical topology to improve energy efficiency and network longevity (Madden et al. 2002; Rajagopalan and Varshney 2006). Amur et al. (2013) study grouped aggregation, focusing on the design and implementation of efficient data structures for batch and streaming computation, though they do not consider staleness, a key performance metric in a geo-distributed setting.

Future Directions for Research

As more and more data is generated by sensors and IoT devices near the edges, research on optimizing geo-distributed streaming analytics can be taken into several directions. One area of research would be to optimize streaming analytics for a heterogeneous, geo-distributed computational environment. This could include edge devices and computing resources, in-networking processing elements, as well as centralized data centers. These elements can differ in their computing, storage, and networking capabilities, and their interactions can lead to interesting resource provisioning and scheduling issues. Another area of research is to achieve desired cost-latency-accuracy trade-offs in streaming analytics by utilizing alternate approaches such as sampling, sketching, as well as compression. For instance, sketching techniques can allow trading off accuracy for latency, and compression techniques can help minimize WAN bandwidth usage. Optimizing different types of queries and multi-query optimization is another area of future research. While multi-query optimization in traditional database systems has focused on minimizing execution time and memory usage, the metrics of interest in a geo-distributed streaming analytics settings are different and will result in new research issues.

References

- Agarwal S et al (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of EuroSys, pp 29–42
- Akida T et al (2013) MillWheel: fault-tolerant stream processing at Internet scale. Proc VLDB Endow 6(11):1033–1044
- Akida T et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc VLDB Endow 8:1792–1803
- Amur H et al (2013) Memory-efficient groupby-aggregate using compressed buffer trees. In: Proceedings of the symposium on cloud computing (SoCC)
- Apache Flink (2016) Scalable batch and stream data processing. <http://flink.apache.org/>
- Apache Storm (2015) Storm, distributed and fault-tolerant realtime computation. <http://storm.apache.org/>
- Beam (2016) Apache Beam (incubating). <http://beam.incubator.apache.org/>
- Boykin O, Ritchie S, O’Connel I, Lin J (2014) Summingbird: a framework for integrating batch and online mapreduce computations. In: Proceedings of VLDB, vol 7, pp 1441–1451
- Chandrasekaran S et al (2003) TelegraphCQ: continuous dataflow processing for an uncertain world. In: Proceedings of the conference on innovative data systems research
- Chen GJ, Wiener JL, Iyer S, Jaiswal A, Lei R, Simha N, Wang W, Wilfong K, Williamson T, Yilmaz S (2016) Realtime data processing at facebook. In: Proceedings of SIGMOD, pp 1087–1098
- Das T, Zhong Y, Stoica I, Shenker S (2014) Adaptive stream processing using dynamic batch sizing. In: Proceedings of the ACM symposium on cloud computing, pp 16:1–16:13
- Flajolet P, Fusy É, Gandouet O et al (2007) HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Proceedings of the international conference on analysis of algorithms
- Gray J et al (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Min Knowl Discov 1(1): 29–53
- Heintz B, Chandra A, Sitaraman RK (2015) Optimizing grouped aggregation in geo-distributed streaming analytics. In: Proceedings of the ACM symposium on high-performance parallel and distributed computing, pp 133–144
- Heintz B, Chandra A, Sitaraman RK (2016a) Trading timeliness and accuracy in geo-distributed streaming analytics. In: Proceedings of the ACM symposium on cloud computing
- Heintz B, Chandra A, Sitaraman RK, Weissman J (2016b) End-to-end optimization for geo-distributed mapreduce. IEEE Trans Cloud Comput 4(3):293–306
- Heintz B, Chandra A, Sitaraman RK (2017) Optimizing timeliness and cost in geo-distributed streaming analytics. IEEE Trans Cloud Comput. <http://ieeexplore.ieee.org/document/8031021/>
- Hwang JH, Cetintemel U, Zdonik S (2008) Fast and highly-available stream processing over wide area networks. In: Proceedings of ICDE, pp 804–813
- Kulkarni S et al (2015) Twitter heron: stream processing at scale. In: Proceedings of SIGMOD, pp 239–250
- Larson PA (2002) Data reduction by partial preaggregation. In: Proceedings of ICDE, pp 706–715
- Madden S, Franklin MJ, Hellerstein JM, Hong W (2002) TAG: a Tiny AGgregation service for ad-hoc sensor networks. In: Proceedings of OSDI, pp 131–146
- Nygren E, Sitaraman RK, Sun J (2010) The Akamai network: a platform for high-performance internet applications. SIGOPS Oper Syst Rev 44(3):2–19
- Peterson L, Anderson T, Culler D, Roscoe T (2003) A blueprint for introducing disruptive technology into the Internet. SIGCOMM Comput Commun Rev 33(1): 59–64
- Pietzuch P et al (2006) Network-aware operator placement for stream-processing systems. In: Proceedings of ICDE
- PlanetLab (2015) <http://planet-lab.org/>
- Podlipnig S, Böszörmenyi L (2003) A survey of web cache replacement strategies. ACM Comput Surv 35(4):374–398
- Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, Stoica I (2015) Low latency geo-distributed data analytics. In: Proceedings of SIGCOMM, pp 421–434
- Qian Z et al (2013) TimeStream: reliable stream computation in the cloud. In: Proceedings of EuroSys, pp 1–14
- Rabkin A, Arye M, Sen S, Pai VS, Freedman MJ (2014) Aggregation and degradation in JetStream: streaming analytics in the wide area. In: Proceedings of NSDI, pp. 275–288
- Rajagopalan R, Varshney P (2006) Data-aggregation techniques in sensor networks: a survey. IEEE Commun Surv Tutor 8(4):48–63
- Vulimiri A, Curino C, Godfrey B, Karanasos K, Varghese G (2015a) WANalytics: analytics for a geo-distributed data-intensive world. In: Proceedings of CIDR
- Vulimiri A, Curino C, Godfrey PB, Jungblut T, Padhye J, Varghese G (2015b) Global analytics in the face of bandwidth and regulatory constraints. In: Proceedings of NSDI, pp 323–336
- Yu Y, Gunda PK, Isard M (2009) Distributed aggregation for data-parallel computing: interfaces and implementations. In: Proceedings of SOSP, pp 247–260
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of SOSP, pp 423–438

Orchestration Tools for Big Data

Saurabh Garg¹, Siqi Wang¹, and Rajiv Ranjan²

¹School of Engineering and ICT, University of Tasmania, Hobart, TAS, Australia

²School of Computing Science, Newcastle University, Newcastleupon Tyne, UK

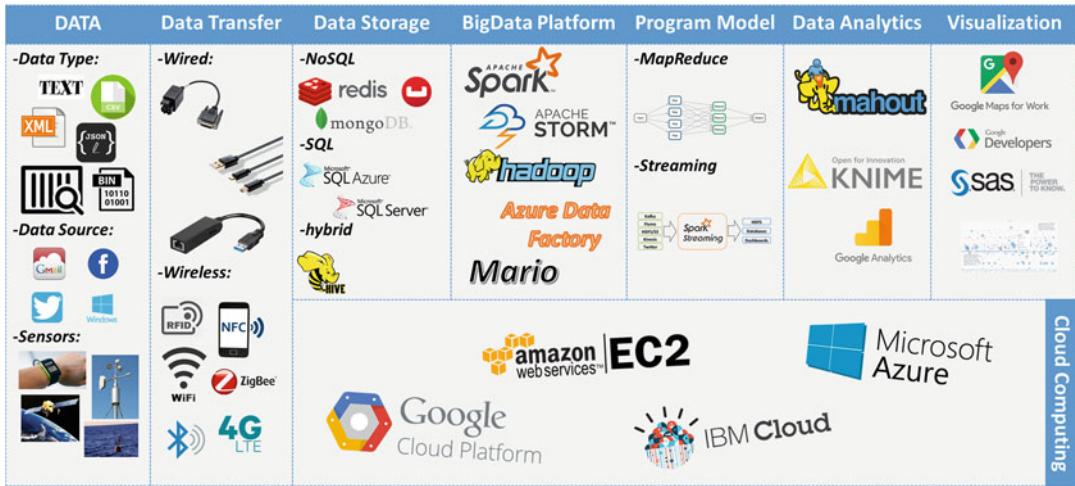
Introduction

With increasing digitization of data and advancement of storage and communication technologies, we are generating quintillion bytes of data every day (Goloboff 1999). According to recent studies, this data collected will become as much as 40 yottabytes by 2020 (Chen et al. 2016). Every second more than 500 millions of tweet messages are posted on Twitter-like social media site. Square Kilometer Array (SKA) radio telescopes can transmit a massive 155.7 terabytes per second. This data explosion is called *big data* that refers to such large quantity of data that it cannot be efficiently handled by traditional data architectures. According to NIST (Grady et al. 2014), big data is characterized by four properties, namely, volume (size of the dataset), variety (data from multiple repositories, domains, or types), velocity (rate of data ingestion), and variability (change in data characteristics). This big data paradigm has brought both opportunities and challenges. Leaders from different organizations such as commercial, academic, and government strongly believe that access to this amount of data is going to revolutionize businesses, scientific research, and government policies.

Big data management and processing is also a challenging task even though it has been partially mitigated by recent advancement in the development of fast processors. However, regardless of how increased CPU speeds are, it cannot surpass the data volumes that are being generated today and will be in future (Kim et al. 2016). To address these issues, there has been a great increase recently in different technologies to sup-

port processing and generating insights from big data. These technologies address different challenges in different stages of data life cycle (Kim et al. 2016) which consist of data collection (includes communication and storage), preparation (involves data cleaning, integration, and organization), analysis (involves extraction of information and knowledge from the data), and action (involves using this knowledge to drive decision-making). Figure 1 shows examples of such technologies such as fast communication technologies (e.g., 5G for mobile data (Chen and Zhao 2014)), data storage (e.g., NoSql databases such as MongoDb (Chodorow 2013), HDFS (Karun and Chitharanjan 2013)), big data platforms (e.g., Hadoop (Vavilapalli et al. 2013) and Spark (Zaharia et al. 2016)), analytics libraries (e.g., Mahout), visualization (e.g., Google Charts), and processing infrastructure (cloud computing).

Based on these technologies, several decision-making applications such as disaster management and intrusion detection can be designed. However, the challenge is how to integrate these technologies in a way that it achieves the objective of users in terms of performance and quality of the solution. This issue or challenge is part of *big data orchestration* process. Big data orchestration process manages whole life cycle of data from collection to insights for the end user. However, user's goals are generally expressed in terms of performance objectives and vary from application to application. For instance, for disaster management applications such as flood modeling, time is a critical performance measure; similarly for some applications such as credit card transaction analysis, a user may want every transaction needs to be processed within a given deadline. For medical data analysis, accuracy of recommendations is the most crucial part. Moreover, data injected might also dynamically vary in generation rate or inherent characteristics, or underline computing infrastructure might fail or degrade in performance. Therefore, big data orchestrator needs to select and configure appropriate tools for each operation, integrate and deploy them to achieve end user goals considering uncertainties



Orchestration Tools for Big Data, Fig. 1 Technologies in big data life cycle

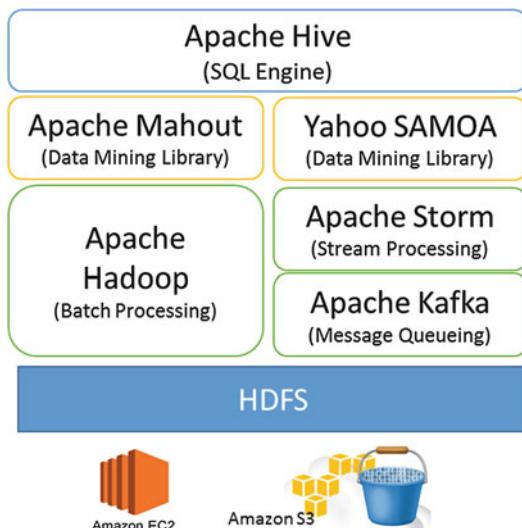
at different levels of big data processing stack. In this chapter, we will discuss in detail big data orchestrator process and its requirements to fulfill users' requirements. We will discuss also the tools and technologies available for enabling orchestration of big data applications. Then we conclude by describing the open challenges.

BigData Orchestration Operations and Challenges

An orchestration platform integrates different tools and technologies for different stages of big data analysis (Ranjan et al. 2017). Figure 1 shows these big data tools in different stages of data starting from data ingestion to visualization of insights. Orchestration platform will select appropriate ones and deploy the whole solution on different cloud infrastructures involving initiating different virtual machines and configuring different services. The orchestration engine configures these tools keeping in consideration end user's specified quality of service (QoS) requirements. It will also establish relationship (data flows) between different tools and services. The orchestration platform also needs to monitor the deployed application's performance and make runtime changes as needed to fulfill QoS requirements. In summary,

there are four key orchestration operations for managing a big data application. To understand these operations and their requirements, let us take an example of flood monitoring and prediction application. This application needs processing of historical and real-time data; thus multiple and heterogeneous big data technologies are required as specified in Fig. 2. The application combines stream data in free text format from Twitter and flood simulation data stored over Amazon S3. In the application, HDFS can be used to store the output from batch (e.g., Hadoop (Karun and Chitharanjan 2013)) and stream processing (e.g., Apache Storm (van der Veen et al. 2015)) platforms. The analytical part of the application can be build using different programming models and libraries such as Mahout. Apache Hive (Huai et al. 2014) can be used to extract insights from database and send these in the form of alerts to the users.

- 1. Big Data Tools Selection:** This is the first stage of orchestration platform where it needs to analyze user's QoS requirements and select appropriate tools. There are several options for each stage of data processing for required solution functionality. The orchestration has to select in a way that different tools (Apache Hadoop, Storm, Spark, Mahout, Hive, MongoDB) from different stages are



Orchestration Tools for Big Data, Fig. 2 Flood monitoring and prediction application

inter-operable and compatible with cloud resources (CPU, storage, network) in addition to user's QoS requirements such as target budget of solution and availability. The hard challenge is to optimally and automatically select tools, hardware, and their configurations such that anticipated QoS constraints (e.g., minimizing event detection delay, minimizing data processing cost) are met while optimizing performance of big data software stack (e.g., minimizing response time for Apache Hadoop) and of cloud resources (maximizing utilization). For example, in Fig. 2, there is a need to optimally select configurations of Apache Hadoop (number of map tasks, number of reduce tasks, map slots per CPU, reduce slots for CPU, max RAM per slot) and Amazon CPU resource (<http://aws.amazon.com/ec2/instance-types>) configurations (I/O capacity, RAM, CPU speed, local storage, cost) driven by application-level performance SLA constraints (analyze 100 GB of Tweets in 10 min subject to maximum budget of \$100). The space of possible configurations for big data processing tools and cloud resource is very large, so computing an optimal solution is NP-complete and therefore intractable given current technology. We need algorithms that

can analyze existing cloud resources and big data processing tools for automatic optimal configuration selection decisions. Minimally, the orchestration platform should provide support for an administration to make these decisions.

2. Big Data Application Deployment (Both Design Time and Runtime): This operation involves instantiating big data tools on selected cloud services with communication protocols and big data database solutions. During deployment, variation in the requirements of each big data software needs to be taken in account. For some big data workloads (high-volume batch processing of historical Tweets by Hadoop in Fig. 2), storage requirements dominate, while for others (transactional query processing by Apache Hive in Fig. 2), computational requirements dominate, and for still others (real-time Twitter stream processing by Apache Storm in Fig. 2), communication requirements dominate. To deploy big data stack on cloud resources which is a multitenancy environment, orchestration systems should deploy the solution in such a way that there will be minimized interference from other application solutions.

3. Monitoring Big Data Application Solution (Runtime): To guarantee QoS requirements, monitoring is a must for making sure each and every tool that is part of the solution is running at optimal performance level. Monitoring QoS objectives of a big data application solution involves regular collection of performance statistics from different deployed big data tools for each stage of data processing cycle and cloud virtual machines. Each stage may have different performance metric for evaluation. For example, at user level or application level, event detection delays and response time are the most important factors, while throughput and latency are important for message queueing systems (e.g., Apache Kafka) and stream processing systems such as Apache Storm. Read/write latency and throughput are important measure for performance of HDFS. Availability, utilization,

and processing speed are essential measure for CPU resources. The challenge in monitoring other than defining the exact performance measure for individual software tools in the solution is integration of them to have unified view of performance for the end user.

4. **Runtime Management:** Based on monitoring data, a big data orchestrator may need to detect the deviation in performance behavior of application and initiate corrective action (adjusting configurations) in a manner that it does not disrupt the availability of solution's output to the end user. An example of corrective action may be to adjust cloud computing resources by adding more number of virtual machines for improving throughput and reduce processing delays. The challenge here comes from unpredictable behavior of ingested data and underline cloud computing resources. As variability is one of the characteristics of big data, it is clear that we cannot expect same data flow rate all the time. If data flow rate is higher, then more computing and storage resources are required. If data flow rate is lower, then hardware resource needs to be minimized; otherwise it will lead to wastage of monetary cost for underutilized resources. The hard challenge in making such adjustment is in computing exact adjustment to make in different parts of software and hardware stack. This requires also accurate prediction models for data flow rates and hardware performance.

Technologies for Big Data Orchestration

In recent years, there are several orchestration tools that have been developed which facilitate orchestration operation to different capacities. In this section we will discuss important orchestration tools. The comparison of these tools is given in Table 1.

Apache YARN

Apache YARN (Vavilapalli et al. 2013) is an open-source resource management technology deployed on Hadoop clusters. YARN supports

allocation of resources for a variety of applications. It runs two daemons, which handle two different tasks: job tracking and progress monitoring. These two daemons are called the resource manager and the application manager, respectively. The resource manager allocates resources to various applications, and the application manager monitors the execution of the process. YARN offers significant advantages in scalability, efficiency, and flexibility compared to the classic MapReduce engine in the first edition of Hadoop; when a user submits an application, an instance of a lightweight process called the ApplicationMaster is started to coordinate the execution of all tasks within the application. This includes monitoring tasks, restarting failed tasks, speculatively running slow tasks, and calculating total values of application counters. In the YARN architecture, a global ResourceManager runs as a master daemon, usually on a dedicated machine, that arbitrates the available cluster resources among various competing applications. The ResourceManager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when.

Apache Mesos

Mesos (Kakadia 2015) is an open-source software originally developed at the University of California at Berkeley. It sits between the application layer and the operating system and makes it easier to deploy and manage applications in large-scale clustered environments more efficiently. Mesos utilizes the modern kernel features in Linux to provide isolation for CPU, memory, I/O, file systems, rack locations, and so on. The important feature of Mesos is that it can integrate several heterogeneous resources and provide easy access to application frameworks. Mesos introduces a distributed two-level scheduling mechanism. Mesos decides how much resources to provide for each big data framework, and the framework determines which resources to accept and which calculations to run on it; it does not set up a large number of server clusters for different parts of the application, but rather

Orchestration Tools for Big Data, Table 1 Comparison of orchestration tools

	Selection	Deployment	Monitoring	Runtime management (reactive)	Runtime management (predictive)
Apache YARN	Manual	Deploys the application on cluster. No resource contention detection or handling	Progress of application execution	Yes (changes in resource consumption and fault tolerance)	Yes
Apache Mesos	Manual	Resource offers are created by Mesos based on specified resource sharing policy such as fair share. No resource contention detection or handling	Monitor failures and resource usage	Fault tolerance	No
AWS Lambda	Automatic	Deploy virtual machines, storage, network, and then applications (written in Java, Node.js, C#, and Python)	Hardware level and software level using CloudWatch	Fault tolerance, scaling of resources	Information not available
MROrchestrator	Automatic	NA	Monitor resource-level bottleneck	Yes (adjust resources based on requirements)	Task's execution time estimation and resource utilization
Azure data factory	Workflow creation interface for users	Deploy the data processing pipeline and cloud resources	Monitor the data flows	Yes, reactive approach by adjusting the resources according to data flow	No
Beygelzimer et al	Automatic	No information available	No information available	Tackle a real-time ectopic beat detection problem in healthcare	No

can share a different part of the resources that can run the different parts of the application without interfering with each other. Mesos also enables dynamically allocation of resources to application frameworks. With the help of Mesos' Docker executor, Mesos can run and manage Docker containers with the Chronos and Marathon frameworks. Docker containers provide a consistent, compact, and flexible way to package applications. Using Docker to implement applications on Mesos provides a truly resilient, efficient, and consistent platform for providing a range of applications in the field or in the cloud. Mesos consists of a master process that manages slave daemons running on each cluster node and frameworks that run tasks on these slaves. The master implements fine-grained

sharing across frameworks using resource offers. Each resource offer is a list of free resources on multiple slaves. When the master node decides to give some resource offers from slaves to application frameworks, they decide whether to accept those resources. The master decides how many resources to offer to each framework according to a given organizational policy, such as fair sharing or strict priority.

Amazon Lambda

AWS Lambda (<https://aws.amazon.com/lambda/details/>) is a serverless compute service that allows the execution of code that is written in Node.js, Python, C#, and Java to be executed in response to events and automatically adjust the underlying compute resources. This service

allows integration with other AWS services and can provide high scalability, performance, and security. AWS Lambda can automatically run code in response to multiple events, such as HTTP requests via Amazon API Gateway, modifications to objects in Amazon S3 buckets, table updates in Amazon DynamoDB, and state transitions in AWS Step Functions. Amazon Lambda selects the appropriate highly available compute infrastructure and performs all the administration needed for its maintenance including automatic scaling, provision, deployment of security patch, monitoring, and logging. Amazon Lambda uses Amazon CloudWatch (<https://aws.amazon.com/cloudwatch/>) to monitor hosted applications which can be used to collect and track metrics, collect and monitor log files, set alerts, and automatically react to changes in your AWS resources.

MROrchestrator

MROrchestrator (Sharma et al. 2012) is a software layer that assumes or requires no changes to the Hadoop framework, making it a simple, flexible, portable, and platform generic design architecture. It provides fine-grained, dynamic, and coordinated allocation of resource to MapReduce jobs in Hadoop. Based on the runtime resource profiles of tasks, MROrchestrator builds online resource estimation models for on-demand allocations; MROrchestrator is complementary to the contemporary resource scheduling managers like Mesos and Next Generation MapReduce (NGM).

Azure Data Factory

Big data requires service that can orchestrate and operationalize processes to refine these enormous stores of raw data into actionable business insights. Azure Data Factory (Klein 2017) is a managed cloud service that's built for these complex hybrid extract-transform-load (ETL), extract-load-transform (ELT), and data integration projects. It is a cloud-based data integration service that allows data-driven workflows in the cloud to be used to coordinate and automate data movement and data

conversion. Using Azure Data Factory, you can create and plan data-driven workflows (called pipelines) that can get data from different data stores. It can process and convert data by using computational services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure machines. In addition, you can publish output data to data storage such as Azure SQL data warehouses for business intelligence (BI) applications. Ultimately, through the Azure data plan, raw data can be organized into meaningful data storage and data lakes to achieve better business decisions.

Azure Data Factory itself does not store any data. It lets you create data-driven workflows to orchestrate the movement of data between supported data stores and the processing of data using compute services in other regions or in an on-premises environment. Azure Data Factory has built-in support for pipeline monitoring via Azure Monitor, API, PowerShell, Microsoft Operations Management Suite, and health panels on the Azure portal. Data factory provides triggers representing the unit of processing that determines when a pipeline execution needs to be kicked off. Azure integration runtime provides the native compute to move data between cloud data stores in a secure, reliable, and high-performance manner. You can set how many data movement units to use on the copy activity, and the compute size of the Azure IR is elastically scaled up accordingly without you having to explicitly adjust the size of the Azure integration runtime. Azure integration runtime supports connecting to data stores and compute services in public network with public accessible endpoints. Use a self-hosted integration runtime for Azure Virtual Network environment.

Beygelzimer et al. (2013)

Beygelzimer et al. proposed an orchestration system for big data that is designed to use a combination of planning and machine learning to automatically determine the most appropriate data-driven workflows to execute in response to a user-specified object; it uses planning and learning system to dynamically determine the most effective analytic workflow to construct and de-

ploy given our computational resources and user-specified task; the system is designed with two major components, a planner and a learner. The purpose of the planner is to discover the set of goals and parameters that match a specific objective such as predicting patient complications and for each such goal to automatically compose and generate code and deploy an analytic workflow that realized the goal. The learner will continuously reevaluate the current mix of analytic workflows deployed and, when deciding to change this set, communicate with the planner which will compose and deploy the analytic workflows, potentially on multiple platforms. In short summary, the planner is running as a set of plugins in the Equinox OSGI container, while the learner component is running as a stream processing application on the IBM InfoSphere Streams platform.

Open Research Issues

Despite several efforts made in orchestrating big data application solutions, still there are significant open research gaps. The following are some of these gaps in each orchestration operation:

- **Big Data Tools Selection:** As we noted in the last section, most big data orchestration platforms such as Apache YARN and Mesos are designed for cluster environments which do not scale on-demand. Moreover, these platforms require application administrator to determine the exact configurations for application execution such as number of map and reduce tasks in case of MapReduce application and also for hardware resources. They are also designed for monolithic applications not for large solutions requiring integration of multiple big data tools and technologies. Moreover, there are almost no calculators except some branded cloud price ones (<http://calculator.s3.amazonaws.com/>, <http://www.windowsazure.com/en-us/pricing/calculator>) that can recommend or compare configurations across big data processing tools and cloud infrastructure

resources based on given QoS performance objectives. In a narrow domain, recent efforts have attempted to automate the configuration selection of Hadoop frameworks over heterogeneous cluster resources such as by Lee and Katz (2011) and AROMA (Lama and Zhou 2012). These works do provide foundation, but future research must focus on developing holistic decision-making frameworks that automate configuration selection across multiple hardware resource types and big data processing frameworks to ensure QoS objectives.

- **Big Data Application Deployment (Both Design Time and Runtime):** As specified before, after selection of right configurations of big data processing systems and cloud infrastructure resources, there are several other factors that need to be taken into account while deployment. In general cloud infrastructure is shared between different applications. Even though virtualization provides some isolation between different applications, it is not possible to exactly divide usage of underline hardware. In context of big data, other than computing resources, network resources also play a key role. Currently in most of cloud provider, it is not possible to preallocate the network resources. It is also important to note that big data orchestration platforms such as YARN, Mesos, and Amazon Lambda have no support for detecting and handling resource contentions. Therefore, new software-based techniques are required that can learn interference in different cloud resources such as CPU, memory, and network.
- **Monitoring Big Data Application Solution (Runtime):** Several monitoring frameworks are available in the literature for monitoring performance of big data processing platforms and cloud. Cluster-wide monitoring frameworks such as Nagios (Josephsen 2007) and Ganglia (Massie et al. 2004) adopted by big data processing platforms such as YARN, Apache Hadoop, and Apache Spark provide information about hardware resource-level metrics (cluster utilization, CPU utilization, memory utilization, and

nature of application: disk-, network-, or CPU-bound). In public cloud computing space, monitoring frameworks (Amazon CloudWatch used by Amazon Lambda, Azure Fabric Controller) typically monitor an entire CPU or VM resource as a black box and so cannot monitor application-level performance metrics. However, still efforts are needed to integrate these monitoring data for whole big data solution (big data processing software and cloud infrastructure) in terms of QoS requirements of the end user. There is also need of mechanisms and tools that can automatically detect any performance degradation based on specified QoS and analyze collected monitoring logs to identify root cause of failure or degradation in the performance.

- **Runtime Management:** The cloud computing community has done extensive research (Ardagna et al. 2014) in developing workload and resource performance modeling techniques for web applications. Public cloud providers provide simplistic reactive approaches to handle peak loads and detecting response time degradation. However, reactive techniques are not efficient solution when a large big data application with multiple software and hardware is needed to be managed. Currently available prediction models cannot be applied directly to the big data workload which has four unique characteristics (volume, velocity, variety, variability). Moreover, there is a need of prediction models for different types of big data-specific performance metrics (batch processing load, stream processing load, data ingestion load, search query processing load) as big data applications differ from multi-tier web applications. To handle dynamic changes to ingested data, failures, or degraded performance of underline hardware, optimal configuration for different components of big data application should be again computed in minimal time. Thus, there is a need of efficient algorithms that can find optimal configurations that need minimal changes in current deployment.

Conclusion

This chapter discussed big data life cycle and tools that are available for different stages of data processing. Given so many tools, libraries, databases, platforms, and infrastructure option, businesses and different organizations need orchestration support for selection of appropriate tool for building their big data application solutions, its deployment, its monitoring, and runtime management of all components of solution and adjustment in hardware resources. The aim of big data orchestration is to ensure that the application solution meets QoS performance objectives, such as availability, throughput, latency, security, cost, and reliability under uncertainties and variability of data and cloud computing resources. We also discuss these orchestration operations in details with current tools that are available in big data context. It is clear from the short survey of these orchestrating platforms that still there are many open challenges that need to be addressed for designing a fully automatic and flexible orchestration platform for big data application solutions.

References

- Ardagna D, Casale G, Ciavotta M, Pérez JF, Wang W (2014) Quality-of-service in cloud computing: modeling techniques and their applications. *J Internet Serv Appl* 5(1):11
- Beygelzimer A, Riabov A, Sow DM, Turaga DS, Udrea O (2013) Big data exploration via automated orchestration of analytic workflows. In: ICAC, pp 153–158
- Chen S, Zhao J (2014) The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication. *IEEE Commun Mag* 52(5):36–43
- Chen X, Li J, Weng J, Ma J, Lou W (2016) Verifiable computation over large database with incremental updates. *IEEE Trans Comput* 65(10):3184–3195
- Chodorow K (2013) MongoDB: the definitive guide: powerful and scalable data storage. O'Reilly Media, Inc., Beijing
- Goloboff PA (1999) Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics* 15(4):415–428
- Grady NW, Underwood M, Roy A, Chang WL (2014) Big data: challenges, practices and technologies: NIST big data public working group workshop at IEEE big data 2014. In: 2014 IEEE international conference on Big Data (Big Data). IEEE, pp 11–15

- Huai Y, Chauhan A, Gates A, Hagleitner G, Hanson EN, O'Malley O, Pandey J, Yuan Y, Lee R, Zhang X (2014) Major technical advancements in Apache Hive. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 1235–1246
- Josephsen D (2007) Building a monitoring infrastructure with Nagios. Prentice Hall PTR, Upper Saddle River
- Kakadia D (2015) Apache Mesos essentials. Packt Publishing Ltd, Birmingham
- Karun AK, Chitharanjan, K (2013) A review on Hadoop—HDFS infrastructure extensions. In: 2013 IEEE conference on information & communication technologies (ICT). IEEE, pp 132–137
- Kim J, Yu SY, Park J (2016) Performance evaluation of multithreaded computations for cpu bounded task. In: 2016 international conference on platform technology and service (PlatCon). IEEE, pp 1–5
- Klein S (2017) Azure data factory. In: IoT solutions in Microsoft's Azure IoT Suite. Springer, Apress, pp 105–122
- Lama P, Zhou X (2012) Aroma: automated resource allocation and configuration of MapReduce environment in the cloud. In: Proceedings of the 9th international conference on autonomic computing. ACM, pp 63–72
- Lee G, Katz RH (2011) Heterogeneity-aware resource allocation and scheduling in the cloud. In: HotCloud
- Massie ML, Chun BN, Culler DE (2004) The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput 30(7):817–840
- Ranjan R, Garg S, Khoskar AR, Solaiman E, James P, Georgakopoulos D (2017) Orchestrating bigdata analysis workflows. IEEE Cloud Comput 4(3):20–28
- Sharma B, Prabhakar R, Lim S-H, Kandemir MT, Das CR (2012) Mroorchestrator: a fine-grained resource orchestration framework for MapReduce clusters. In: 2012 IEEE 5th international conference on cloud computing (CLOUD). IEEE, pp 1–8
- van der Veen JS, van der Waaaij B, Lazovik E, Wijbrandi W, Meijer RJ (2015) Dynamically scaling Apache storm for the analysis of streaming data. In: 2015 IEEE first international conference on big data computing service and applications (BigDataService). IEEE, pp 154–161
- Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S et al (2013) Apache Hadoop YARN: Yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. ACM, p 5
- Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ et al (2016) Apache Spark: a unified engine for Big Data processing. Commun ACM 59(11):56–65

Overview of Online Machine Learning in Big Data Streams

- Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning

P

Parallel Graph Processing

Da Yan¹ and Hang Liu²

¹Department of Computer Science,

The University of Alabama at Birmingham,
Birmingham, AL, USA

²University of Massachusetts Lowell, Lowell,
MA, USA

Definitions

The term *parallel graph processing* refers to the use of multiple cores to process a graph for the purpose of (1) speeding up of processing and (2) scaling to bigger graphs. The environment can be (1) a stand-alone machine running multiple threads or (2) a distributed cluster of machines (i.e., the shared-nothing architecture).

Overview

Modern big graph processing systems place emphasis on two aspects:

1. user-friendliness: the programming interface should be designed based on an intuitive computation model, to allow algorithm developers to focus on the graph analytics logic without touching low-level execution details (e.g., network communication);

2. efficiency: the underlying execution engine should guarantee high-throughput execution and automatically support fault tolerance and horizontal/vertical scaling.

Since comprehensive surveys of this area already exist (Yan et al. 2017a,d), this chapter aims at a succinct and more up-to-date review of the key programming paradigms and systems for big graph analytics, which are commented based on (1) their computation models and target applications and (2) their execution environments.

We remark that the focus of this chapter is on graph computation, not on graph storage, esp. for distributed systems running with a distributed shared-nothing cluster. For these systems, graph storage is an orthogonal topic, since it can load data from any distributed store, such as Hadoop Distributed File System (HDFS) or even a distributed graph database like TITAN. W.l.o.g, we assume that graph data are stored on HDFS, which provides “parallel IO” and “fault tolerance”: HDFS stores a big input file as blocks of size 64MB each, and each block is replicated on three machines to be tolerant to machine failures. A distributed system loads the data from HDFS in parallel with each machine reads only part of the input data, so that all disks are utilized to achieve an aggregate bandwidth.

We also remark that a distributed system is not necessarily faster than a stand-alone solution (esp. for small graphs). In fact, for data-intensive graph analytic tasks where the data volume to transmit is comparable to the input

graph size, network communication is usually the performance bottleneck. With N machines, each machine only needs to perform computation over about $1/N$ of the vertices, but needs to pay for the additional communication cost incurred by these vertices. In other words, a distributed system is only beneficial if the number of machines is large enough to outweigh the incurred communication overhead.

This chapter uses the following graph notations. Given a graph $G = (V, E)$, we denote the number of vertices $|V|$ by n and the number of edges $|E|$ by m . For an undirected graph, we denote the set of *neighbors* of a vertex v by $\Gamma(v)$ and the *degree* of v by $d(v) = |\Gamma(v)|$. For a directed graph, we denote the set of *in-neighbors* and *out-neighbors* of v by $\Gamma_{in}(v)$ and $\Gamma_{out}(v)$ and the *in-degree* and *out-degree* of v by $d_{in}(v) = |\Gamma_{in}(v)|$ and $d_{out}(v) = |\Gamma_{out}(v)|$, respectively.

Key Research Findings

We now review the three key computing paradigms for big graph analytics, i.e., two vertex-centric paradigms (Pregel-like message passing v.s. shared-memory abstraction) and a subgraph-centric paradigm.

Pregel-Like Systems

Malewicz et al. (2010) pioneered the “think-like-a-vertex” computing model where users develop a distributed graph algorithm simply by specifying the behavior of a generic vertex. Their system, Google’s Pregel, is designed based on the bulk synchronous parallel (BSP) model. It distributes vertices to different machines in a cluster, where each vertex v is associated with its adjacency list. A program in Pregel implements a user-defined function (UDF) *compute()* and proceeds in iterations (called *supersteps*). In each superstep, the program calls *compute(msgs)* for each “active” vertex v to perform the user-specified computation logic on v , such as processing v ’s incoming messages *msgs* (sent in the previous superstep), sending messages to other vertices (to be received in the next superstep), and making

v vote to halt. A halted vertex is reactivated if it receives a message in a subsequent superstep. The program terminates when all vertices vote to halt, and there is no pending message for the next superstep.

Pregel numbers the supersteps so that a user may access the current superstep number in *compute()*. This allows a Pregel algorithm to perform different operations in different supersteps.

Pregel allows users to implement a *combine()* function, which specifies how to combine messages that are to be sent from a machine M_i to the same vertex v in a machine M_j . These messages are combined into a single message, which is then sent from M_i to v in M_j . Combiner is applied only when messages can be aggregated with a commutative and associative operation.

Pregel also supports global communication through “aggregator.” Each vertex can provide a value to an aggregator in *compute()* in a superstep. The system aggregates those values and makes the aggregated result available to all vertices in the next superstep.

We now look at two examples on how to write a vertex-centric program.

Example 1 We present the Hash-Min algorithm (Yan et al. 2015) for computing connected components (CCs) in an undirected graph. Each vertex v maintains the smallest vertex ID it has seen, denoted by $\min(v)$. Hash-Min lets each vertex broadcast the smallest vertex ID seen so far, and when the algorithm terminates, $\min(v)$ equals the smallest vertex ID in v ’s CC.

The algorithm works as follows. In superstep 1, each vertex v sets $\min(v)$ as v ’s ID, broadcasts $\min(v)$ to all its neighbors, and votes to halt. In superstep i ($i > 1$), each vertex v receives messages from its neighbors; let \min^* be the smallest ID received, if $\min^* < \min(v)$, v sets $\min(v)$ to be \min^* and broadcasts \min^* to its neighbors. All vertices vote to halt at the end of a superstep. When the process converges, all vertices are halted, and vertices with the same value of $\min(v)$ constitute a CC. □

In this algorithm, message combining with the MIN operator can be applied to reduce commun-

nication. Specifically, since $compute()$ only cares about the minimum ID received, messages to be sent from a machine toward the same vertex v can be combined into one message that equals their minimum, before being sent to v . However, we remark that message combining is not applicable to an arbitrary Pregel algorithm, and many other Pregel-like systems (Zhou et al. 2014; Zhang et al. 2014) are built on the assumption that message combining is always applicable (to enable more optimization opportunities) but restricts the expressiveness compared with Pregel.

In Pregel, a vertex v does not have to communicate only with its direct neighbor in $\Gamma(v)$ (as in Hash-Min); it can send messages to any vertex u as long as u 's ID is known. Pregel is very suitable to implement parallel algorithms that adopt the pointer jumping (a.k.a. path doubling) technique to achieve good computational complexity. In such an algorithm, a vertex v may communicate with a vertex that is not in $\Gamma(v)$. Consider the following *list ranking* problem, which is a building block of the Euler tour technique (e.g., useful for computing biconnected components).

Example 2 Consider a linked list \mathcal{L} with n objects, where each object v is associated with a value $val(v)$ and a link to its predecessor $pred(v)$. The object v at the head of \mathcal{L} has $pred(v) = null$. For each object v in \mathcal{L} , the *list ranking* problem computes the summed value of v , denoted by $sum(v)$, as the sum of the values of all the objects from v following the predecessor link to the head.

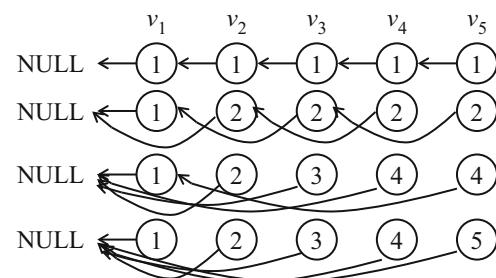
In list ranking, objects in \mathcal{L} are given in arbitrary order. A logical view of \mathcal{L} is simply a directed graph consisting of a single simple path. We now present the Pregel algorithm. Initially, each vertex v sets $sum(v)$ to be $val(v)$. Then in each round, each vertex v does the following: If $pred(v) \neq null$, v sets $sum(v) \leftarrow sum(v) + sum(pred(v))$ and $pred(v) \leftarrow pred(pred(v))$; otherwise, v votes to halt. Note that the if-branch can be accomplished in three steps: (1) v sends a message containing its ID to $u = pred(v)$ requesting for the values of $sum(u)$ and $pred(u)$; (2) u sends back the requested values to v ; and

(3) v updates $sum(v)$ and $pred(v)$. This process repeats until $pred(v) = null$ for every vertex v , at which point all vertices vote to halt and we have $sum(v)$ as desired. \square

Figure 1 illustrates how the algorithm works. Initially, objects v_1-v_5 form a linked list with $sum(v_i) = val(v_i) = 1$ and $pred(v_i) = v_{i-1}$. Let us now focus on v_5 . In Round 1, we have $pred(v_5) = v_4$ and we set $sum(v_5) \leftarrow sum(v_5) + sum(v_4) = 1 + 1 = 2$ and $pred(v_5) \leftarrow pred(v_4) = v_3$. One can verify the states of the other vertices similarly. In Round 2, we have $pred(v_5) = v_3$, and we set $sum(v_5) \leftarrow sum(v_5) + sum(v_3) = 2 + 2 = 4$ and $pred(v_5) \leftarrow pred(v_3) = v_1$. In Round 3, we have $pred(v_5) = v_1$, and we set $sum(v_5) \leftarrow sum(v_5) + sum(v_1) = 4 + 1 = 5$ and $pred(v_5) \leftarrow pred(v_1) = null$. Obviously, the algorithm stops in $O(\log n)$ rounds, and each object v sends (resp. receives) at most one message in each round.

List ranking is yet one of the many fundamental graph problems (including graph connectivity) that can be solved with Pregel using linear cost per superstep and at most $O(\log n)$ supersteps. These Pregel algorithms are scalable and have been studied in Yan et al. (2014b). In contrast, the Hash-Min algorithm can take up to $(|V| - 1)$ supersteps.

Optimizations. Since Pregel is only for internal use in Google, many open-source Pregel-like systems have emerged with various improvements. Giraph (Ching et al. 2015) (resp. Pregel+ Yan et al. 2015) provides a stable



Parallel Graph Processing, Fig. 1 Illustration of list ranking

open-source Pregel implementation in Java (resp. C++). GraphD (Yan et al. 2017c) lets each machine stream the adjacency lists of its assigned vertices, as well as messages toward and generated by those vertices, on local disks to eliminate the memory requirement. There is no sacrifice in performance when the network communication is the bottleneck (as typical to HDD+Ethernet and SSD+Infiniband settings) since disk streaming is overlapped with and covered by the message transmission.

GPS (Salihoglu and Widom 2013) constructs mirrors of each high-degree vertex v in other machines, so that v only needs to send its updated value to its mirror at a machine M which then forwards this update locally to v 's neighbors on M (rather than sending a message to each neighbor on M). Yan et al. (2015) further discover that there is a trade-off between vertex mirroring and message combining (in terms of reducing the message number) and provide analytical results to help determine which vertices should (and should not) be mirrored. The work also provides another technique to avoid the communication bottleneck caused by pointer jumping.

In many analytic tasks, graph computation is just one sub-step among a workflow of heterogeneous operations (e.g., MapReduce for preprocessing raw data into a graph). If different systems are used for different sub-steps, the output from one system has to be written to HDFS and then loaded by another system. General-purpose Big Data systems attempt to maintain a consistent data representation, such as the “RDD” of Apache Spark, to eliminate the data export/import cost between operations. To allow user-friendly programming of graph analytics, these systems are extended with a vertex-centric API, such as Spark’s GraphX (Gonzalez et al. 2014) and Flink’s Gelly (Carbone et al. 2015). Their aim is to achieve less end-to-end processing time by eliminating data exports/imports between consecutive operations.

Block-Centric Paradigm. The vertex-centric model passes updates slowly (one hop per superstep), and almost any communication along an edge requires network transmission. To

overcome this weakness, several systems propose to partition a big graph first and then let messages propagate among partitions (rather than vertices); updates propagated within each partition do not incur network communication.

Giraph++ (Tian et al. 2013) pioneers this idea by letting users define *compute()* over a partition of vertices (rather than each individual vertex). Blogel (Yan et al. 2014a) further allows users to associate status (e.g., values) with each partition (a connected subgraph called block) and allows both a vertex and a block to call a UDF *compute()*. With Blogel, the Hash-Min algorithm can just let blocks maintain and exchange the minimum block ID seen, without any vertex-level processing.

A key issue in the block-centric model is how to track vertices. In Pregel, the machine of a vertex v can be computed through a hash function over v 's ID, and it is called whenever a message needs to be sent to a vertex v . After graph partitioning, vertices in a block should be distributed to the same machine, but it is difficult to find a hash function to capture the vertex-to-machine mapping. Giraph recodes vertex IDs so that their ID-to-machine mapping can be captured with a hash function, while Blogel simply let each vertex ID to comprise the ID of the machine where the vertex locates. Giraph partitions a graph using a METIS-like approach, where METIS (Karypis and Kumar 1998) is classical graph partitioning algorithm with a high partitioning quality but limited scalability. Blogel provides several highly scalable partitioning tools including a general-purpose one based on the idea of graph Voronoi diagram, though the partitioning quality cannot match that of METIS.

Fan et al. (2017) propose the GRAPE system that only requires users to provide sequential graph algorithms; GRAPE uses METIS to partition a graph into high-quality fragments, and since only a small fraction of edges cross different machines, they are tracked by a master machine to decide the direction of message passing. However, the efficiency of this approach depends highly on the partitioning quality, and the sequential algorithms should meet a “monotonic condition.”

Shared-Memory Abstraction

Unlike Pregel where a vertex pushes its update to its neighbors, another kind of vertex-centric paradigm is shared-memory abstraction, where a vertex directly accesses the data of its adjacent vertices and edges during computation. However, these systems just simulate a shared-memory environment, and the actual execution is not shared memory but rather (1) distributed or (2) out of core. In fact, the simulation comes with some costs in both performance (shall be discussed shortly) and expressiveness (e.g., a vertex cannot communicate with a non-neighbor like in Pregel).

GraphLab. GraphLab (Gonzalez et al. 2012) pioneered the shared-memory abstraction in a distributed environment. It partitions edges (rather than vertices) among machines so that the workloads of processing a high-degree vertex can be distributed among multiple machines. However, if an edge (u, v) is distributed to a machine M , M needs to keep the states of both u and v ; therefore, the state of a vertex v is replicated at all machines that keep adjacent edges of v . This adds additional complication in value synchronization among vertex replicas, and it also uses more memory than Pregel. To reduce the number of replicas, GraphLab adopts a greedy strategy to distribute edges to machines, so that an edge (u, v) is more likely to be assigned to a machine that already maintain the states of u and/or v .

GraphLab adopts a Gather-Apply-Scatter (GAS) computation model, where a vertex v first gathers values along adjacent edges, aggregates these values to compute v 's new value, and then scatters value updates along adjacent edges. The model expressiveness is similar to Pregel algorithms where message combining is applicable. However, GraphLab supports asynchronous execution, which is desirable for graph problems where vertex values converge asymmetrically: vertices that converge more slowly can compute for more iterations. For other algorithms, GraphLab's synchronous mode is even faster as was observed by Lu et al. (2014), as asynchronous execution

incurs locking/unlocking cost. Moreover, the result of GraphLab is often approximate since a vertex is considered as converged and stops computation (and thus scattering its update) when the difference of value update is small. Zhang et al. (2014) provide a better solution that guarantees result exactness with synchronous execution, and asymmetric convergence is solved by selecting vertices with top- k largest difference of value update to perform computation in each iteration.

Stand-alone Systems. Shared-memory abstraction is a natural and preferred choice for a stand-alone system (i.e., a system running with a single machine), since there is no network overhead, and faster convergence is achieved through immediate access to the most up-to-date vertex values. Stand-alone systems perform out-of-core graph processing to eliminate the need of huge memory space: they partition vertices according to disjoint ranges of vertex ID and load one vertex partition to memory at a time for processing. The programming models are also just variants of GAS. GraphChi (Kyrola et al. 2012) needs to load all vertices in a partition, along with all their adjacent edges, into memory before processing of the partition begins; it smartly stores the adjacent edges of the vertex partitions (into the so-called shards) so that edge reads/writes are as sequential as possible to maximize disk bandwidth. X-Stream (Roy et al. 2013) only loads all vertices in a partition into memory, while edges are streamed on local disk(s); the model eliminates the need of ordering edges which is required by GraphChi's sharding.

Many other stand-alone graph processing systems have been developed. A number of them feature the efficient execution with new hardware (e.g., SSDs, GPUs). For example, Liu and Huang (2017) developed the Graphene system to fully utilize the bandwidth of an array of SSDs when performing external-memory graph processing; the system hides the details of IO processing from end users. Due to space limit, we refer interested readers to Chaps. 5.2, 9, and 10 of Yan et al. (2017a) for detailed models of the stand-alone systems.

Subgraph-Centric Paradigm

Existing research on parallel graph processing is dominant on problems with low computational complexity, such as iterative operations like random walks and graph traversals. Recall that Yan et al. (2014b) indicate that a Pregel algorithm is scalable if it has $O(n + m)$ cost per iteration and runs for at most $O(\log n)$ iterations, i.e., a complexity bounded by $O((n + m) \log n)$. However, there are many graph mining problems (e.g., community detection, subgraph matching) with a high complexity: the workloads are caused by the huge search space (i.e., subgraphs of a big graph), which can be much larger than the size of the input graph. We call these problems compute-intensive, in contrast to vertex-centric programs that are data-intensive with CPUs underutilized.

In a compute-intensive problem, network communication is no longer the bottleneck, and the goal of a distributed system is to keep the CPUs busy. Since the output of a graph mining problem is defined over subgraphs, a subgraph-centric API is more natural than a vertex-centric one. Some recent works attempt to perform subgraph mining by first constructing subgraph candidates, but the graph construction phase is communication-intensive and CPUs are underutilized. See Chapter 7 of Yan et al. (2017a) for the details.

A satisfactory solution is given by the G-thinker system (Yan et al. 2017b), which decomposes the computation over a big graph G into computations over (possibly overlapping) subgraphs of G that are much smaller, such that each result subgraph can be found in exactly one decomposed subgraph. If a decomposed subgraph is still too big, it can be further decomposed so that the resulting decomposed subgraphs can be distributed to different machines for balanced parallel processing.

We illustrate this divide-and-conquer idea by considering maximal clique enumeration. We decompose the computation over a big graph $G = (V, E)$ into that over a set of G 's subgraphs G_1, G_2, \dots, G_n , where G_i is constructed by expanding from a vertex $v_i \in V$. Let us define $\Gamma_{gt}(v) = \{u \in \Gamma(v) | u > v\}$ where vertices are compared according to their IDs. If we construct

G_i as the subgraph induced by $\{v_i\} \cup \Gamma(v_i)$ (called v_i 's 1-ego network), then we can find all cliques from these 1-ego networks since any two vertices in a clique must be neighbors of each other. To avoid redundant computation, we redefine G_i as induced by $\{v_i\} \cup \Gamma_{gt}(v_i)$, i.e., G_i does not contain any neighbor $v_j < v_i$. This is because any clique containing both v_i and v_j has already been computed when processing G_j . Obviously, any clique C (let the smallest vertex in C be v_i) is only computed once, i.e., when G_i is processed. If v_i has a very high degree, G_i is big which incurs heavy computation workload, and we can recursively generate decomposed subgraphs on G_i from neighbors of v_i , similarly as how we generate the seeding decomposed subgraphs on the original graph G , but conditioned on the fact that v_i is already included in any clique subsequently found.

G-thinker Design. G-thinker performs computation on subgraphs. Each subgraph g is associated with a task, which performs computation on g and grows g by pulling adjacent vertices/edges when needed. Tasks are spawned from individual vertices. Different tasks are independent, while each task performs computation iteratively so that if it is waiting for responses of its data requests, it will be hanged up in a task queue to allow CPUs to process other tasks.

G-thinker loads an input graph by partitioning vertices (along with their adjacency lists) among the memory of different machines, forming a distributed key-value store that returns $\Gamma(v)$ for every request to pull vertex v . Each machine spawns tasks from its loaded vertices, but idle machines may steal tasks from other machines for load balancing. Vertex requests and responses are sent in batches to amortize the round-trip time, and multiple requests for the same vertex v from a machine are merged into one request before sending. The received remote vertices are put in a local vertex cache to allow sharing by all local tasks. The vertex cache maintained by a storage manager is in-memory and has a limited capacity. When a vertex is no longer locked by any local task, it can be deleted to make room for new vertices requested by other tasks.

Users write a G-thinker program by implementing two UDFs. The first is *compute(frontier)*, which specifies how a task computes for one iteration. If $t.\text{compute}()$ returns true, task t needs to be processed by more iterations; otherwise, t is finished after the current iteration. Inside $t.\text{compute}()$, a user may access and update t 's subgraph g and t 's state and call *pull(u)* to request vertex $\Gamma(u)$ for use in t 's next iteration. Here, u is usually in the adjacency list of a previously pulled vertex (tracked by the function input *frontier*), and *pull(u)* expands the frontier of g (usually in a breadth-first manner). A user may also call *add_task(task)* in $t.\text{compute}()$ to add a newly created task to the task queue, which is useful when g is too large and needs further decomposition.

The second UDF is *task_spawn(v)*, which specifies how to create tasks according to a vertex v . Inside the function, users may create 0 or more tasks by examining v 's attribute and adjacency list and add them to the task queue by calling *add_task()*. G-thinker calls this function on every vertex to spawn tasks.

Example of Applications

Vertex-centric algorithms have been developed to solve fundamental graph problems such as breadth-first search, list ranking, spanning tree, Euler tour, pre/post-order traversal, connected components, biconnected components, strongly connected components, etc. (Yan et al. 2014b). It has also been applied to various real applications such as social network analysis (Quick et al. 2012) and genome assembly (Yan et al. 2018).

Subgraph-centric frameworks have been applied for problems including subgraph-structure (triangle, clique, quasi-clique) mining and graph matching (Yan et al. 2017b), motif counting in biological networks, finding social circles, and personalized recommendations (Quamar et al. 2016).

Future Directions for Research

One promising direction of research is the subgraph-centric paradigm for compute-intensive

(and/or recursive) graph analytics, which is still in its infancy. Existing research on graph processing systems dominantly focuses on optimizing data-intensive iterative graph analytics, limiting their applicability.

Another interesting research area is to consider how to optimize vertex-centric and graph-centric workloads on new hardware such as SSDs and GPUs. While stand-alone solutions have been proposed, it would be interesting to see distributed solutions that utilize a cluster of machines equipped with new hardware. This would pose more challenges since the gap between slow data moving (i.e., network communication) and faster computation (e.g., with GPUs) becomes larger.

Finally, it would be interesting to see more applications developed on these big graph processing frameworks. Current research in this area are dominantly on system design and optimizations, and papers discussing applications of these systems are still very limited in number.

Cross-References

- ▶ Big Data Architectures
- ▶ Graph Data Management Systems
- ▶ Graph Processing Frameworks
- ▶ Large-Scale Graph Processing System
- ▶ Parallel Processing with Big Data
- ▶ Scalable Architectures for Big Data Analysis

References

- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink™: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38
- Ching A, Edunov S, Kabiljo M, Logothetis D, Muthukrishnan S (2015) One trillion edges: graph processing at facebook-scale. PVLDB 8(12):1804–1815
- Fan W, Xu J, Wu Y, Yu W, Jiang J, Zheng Z, Zhang B, Cao Y, Tian C (2017) Parallelizing sequential graph computations. In: SIGMOD, pp 495–510
- Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) Powergraph: distributed graph-parallel computation on natural graphs. In: OSDI, pp 17–30
- Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. In: OSDI, pp 599–613

- Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput* 20(1):359–392
- Kyrola A, Blelloch GE, Guestrin C (2012) GraphChi: Large-scale graph computation on just a PC. In: OSDI, pp 31–46
- Liu H, Huang HH (2017) Graphene: fine-grained IO management for graph computing. In: FAST, pp 285–300
- Lu Y, Cheng J, Yan D, Wu H (2014) Large-scale distributed graph computing systems: an experimental evaluation. *VLDB J* 8(3):281–292
- Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: SIGMOD, pp 135–146
- Quamar A, Deshpande A, Lin JJ (2016) Nscale: neighborhood-centric large-scale graph analytics in the cloud. *VLDB J* 25(2):125–150
- Quick L, Wilkinson P, Hardcastle D (2012) Using pregel-like large scale graph processing frameworks for social network analysis. In: International conference on advances in social networks analysis and mining, ASONAM 2012, Istanbul, pp 457–463
- Roy A, Mihailovic I, Zwaenepoel W (2013) X-stream: edge-centric graph processing using streaming partitions. In: SOSP, pp 472–488
- Salihoglu S, Widom J (2013) GPS: a graph processing system. In: SSDBM, pp 22:1–22:12
- Tian Y, Balmin A, Corsten SA, Tatikonda S, McPherson J (2013) From “think like a vertex” to “think like a graph”. *VLDB J* 7(3):193–204
- Yan D, Cheng J, Lu Y, Ng W (2014a) Blobel: a block-centric framework for distributed computation on real-world graphs. *VLDB J* 7(14):1981–1992
- Yan D, Cheng J, Xing K, Lu Y, Ng W, Bu Y (2014b) Pregel algorithms for graph connectivity problems with performance guarantees. *VLDB J* 7(14):1821–1832
- Yan D, Cheng J, Lu Y, Ng W (2015) Effective techniques for message reduction and load balancing in distributed graph computation. In: WWW, pp 1307–1317
- Yan D, Bu Y, Tian Y, Deshpande A (2017a) Big graph analytics platforms. *Found Trends Databases* 7(1–2): 1–195. <https://doi.org/10.1561/1900000056>
- Yan D, Chen H, Cheng J, Özsu MT, Zhang Q, Lui JCS (2017b) G-thinker: big graph mining made easier and faster. *CoRR* abs/1709.03110
- Yan D, Huang Y, Liu M, Chen H, Cheng J, Wu H, Zhang C (2017c) GraphD: distributed vertex-centric graph processing beyond the memory limit. *IEEE Trans Parallel Distrib Syst* 29(1):99–114
- Yan D, Tian Y, Cheng J (2017d) Systems for big graph analytics. Springer briefs in computer science. Springer, Cham. <https://doi.org/10.1007/978-3-319-58217-7>
- Yan D, Chen H, Cheng J, Cai Z, Shao B (2018) Scalable de novo genome assembly using pregel. In: ICDE
- Zhang Y, Gao Q, Gao L, Wang C (2014) Maiter: an asynchronous graph processing framework for delta-based accumulative iterative computation. *IEEE Trans Parallel Distrib Syst* 25(8):2091–2100
- Zhou C, Gao J, Sun B, Yu JX (2014) Mograph: scalable distributed graph processing using message online computing. *VLDB* 8(4):377–388

Parallel Join Algorithms in MapReduce

Spyros Blanas

Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Definitions

The MapReduce framework is often used to analyze large volumes of unstructured and semi-structured data. A common analysis pattern involves combining a massive file that describes events (commonly in the form of a log) with much smaller reference datasets. This analytical operation corresponds to a parallel join. Parallel joins have been extensively studied in data management research, and many algorithms are tailored to take advantage of interesting properties of the input or the analysis in a relational database management system. However, the MapReduce framework was designed to operate on a single input and is a cumbersome framework for join processing. As a consequence, a new class of parallel join algorithms has been designed, implemented, and optimized specifically for the MapReduce framework.

Overview

Since its introduction, the MapReduce framework (Dean and Ghemawat 2004) has become extremely popular for analyzing large datasets. The success of MapReduce stems from hiding details of parallelization, fault tolerance, and load balancing over a generic programming abstraction. One frequent data analysis pattern in MapReduce starts with a log of events, such as a click stream, a log of phone call records, or even a sequence of transactions in flat files. MapRe-

duce is then used to combine information in the log with other reference data to compute statistics and derive business insights from the data. This analytical pattern corresponds to a parallel join.

A key reason of the popularity of MapReduce for this task over a traditional database system is the sheer volume of data involved in a join, which can be hundreds of terabytes (Blanas et al. 2010). Merely formatting and loading that much data into a database system in a timely manner is a challenge. In addition, the log records do not always follow the same schema, or the interpretation of a log record may change over time. MapReduce gives users the flexibility to write custom code to parse attributes of interest.

Key Research Findings

It turns out it is possible – but not always straightforward – to implement well-known join strategies in MapReduce. Many join algorithms, in fact, do not require any modification to the basic MapReduce framework and thus can be readily used by existing MapReduce platforms like (Hadoop 2017).

Programming in MapReduce involves defining appropriate *map* and a *reduce* function for the specific analysis task at hand. The map function takes a key-value pair (k, v) as input from a distributed file system and generates one or more pairs of (k^*, v^*) as the output. Each map task processes a non-overlapping chunk of the input file called a *split*. Typically, the size of a split is the block size of the distributed file system. The *reduce* function takes as the input a pair of $(k^*, list(v^*))$, where $list(v^*)$ is a list of all v^* values with key k^* . The reduce function produces yet another key-value pair as the output which is stored in a distributed file system.

Repartition Join

We consider an equi-join between a reference table R and an event log S on a single column. In general, R is much smaller than S . The standard repartition join shuffles data in both R and S such that records with identical join keys are colocated

in the same node and then processes the join locally.

The repartition join can be implemented in a single MapReduce job. In the map phase, each map task works on a split of either R or S . The map function tags the record with its originating table to identify which table an input record is from and creates a composite key of the form $(join\ key, tag)$. The partitioning function is customized so that the hash is computed from just the join key part of the composite key. This way all records with the same join key will be assigned to the same reduce task.

For each join key, the reduce function first separates the input records into two sets according to the table tag. Once the entire input has been split, the repartition algorithm then performs a cross-product between records in these sets.

One optimization is to ensure that the table tags are generated such that the tag for table R lexicographically precedes the tag for table S . This way, the reducer that processes a given join key will see all R records for this join key first. Hence, the reducer can only buffer records from table R and start producing data as soon as the first record from table S is encountered.

P

Broadcast Join

In many instances, the event log S is orders of magnitude larger than the reference table R . Instead of moving both R and S across the network, one can broadcast the smaller table R to all nodes. In a MapReduce context, broadcasting is performed via the distributed file system when all tasks read the same file(s).

A broadcast join can be run as a MapReduce job with no reduce function. (The broadcast join is sometimes referred to as a map-side join because of this property.) Each map task first reads R and builds an in-memory hash table on the join key. Then every map task processes a split of S and probes the hash table with the join key of each record. If the key matches, the two records are joined. The map task thus computes the broadcast join and writes directly to the distributed file system without invoking a reduce function. This avoids sorting both tables and,

more importantly, avoids the network overhead for moving the entire log S in the reduce phase.

The broadcast join is amenable to two optimizations. The first is to dynamically pick the side to build the hash table. The motivating observation is that the default split size (commonly 128 MB) may mean that R is larger than the fraction of L that will be processed by this map task. A common optimization is to build the hash table on the fraction of L and use R to probe into this hash table instead.

The second optimization is to materialize the hash table in the local node. A naive implementation of the broadcast join would reread R from the distributed file system for every map task. Because a node processes multiple map tasks, this means retransmitting R multiple times during join processing. Instead, one can serialize the hash table R in the local file system and redirect subsequent map tasks to the local copy. These temporary files need to be cleared after the job terminates. This amortizes the transmission and processing overhead of R over all the map task invocations in this node.

Semi-join

The repartition and broadcast algorithms transfer all records in R ; although many they may not be referenced by any records in S . For instance, many users of a social network may not have been active in the last hour. The semi-join algorithm avoids reading and manipulating records in R that would not match any record in the log S , but it does this at the cost of an extra scan of S .

A semi-join is implemented as three separate MapReduce jobs that run sequentially. The first phase of the semi-join collects all the unique keys in S . This corresponds to using a reduce function that only emits the join key of each group. An implementation may also aggregate in the map function by building a main-memory hash table to determine the set of unique join keys per split of S . Aggregating in the map function curtails duplicate keys when they appear in the same split in S . This optimization is particularly effective when the actions that are recorded in the log exhibit temporal locality. The second phase of the

semi-join is a broadcast join between the unique keys in S (from phase one) with R to produce the exact subset $R' \subset R$ that is necessary to compute the join. The third and final phase of the semi-join is broadcasting R' (from phase two) and joining with S to produce the join output on the distributed file system.

The semi-join procedure described above ensures that every record in R' matches at least one record in S . However, the third phase of the semi-join will still broadcast unnecessary records for the given split of S that each map task will process. One optimization is to generate a subset R'_i per split S_i of S . This makes the final phase of the semi-join cheaper, since it moves exactly the records that will match in the current split of S . This optimization must be applied judiciously as the first two phases become significantly more involved.

Theta Joins

Many join operations specify join conditions other than key equality. Yet, the reduce function of the MapReduce framework operates on records with identical keys. This design constraint makes it challenging to perform joins with arbitrary join conditions in MapReduce. Okcan and Riedewald have introduced a randomized algorithm, called 1- Bucket-Theta, that evaluates arbitrary joins in a single MapReduce job (Okcan and Riedewald 2011).

The motivating observation is that one can compute arbitrary join conditions in MapReduce inside the reduce function, if each invocation receives a superset of all matching tuples. The question then becomes what reduce key should the map function produce to accomplish this. One naive strategy is to assign every tuple to a single reduce key. The reduce function will compute the correct result, but this strategy does not utilize the parallelism of the MapReduce framework. Another naive strategy is to construct a map function that emits a set (k_i, v) tuples per input record v , where k_i enumerates every key in the key domain. This naive solution is theoretically sound and scalable but infeasible in practice

due to the cost of enumerating the entire key domain.

An efficient theta join algorithm should aim to emit as few reduce keys as possible but also balance the work between reducers. Okcan and Riedewald cast this problem as an optimization problem: given a join matrix between all R and S tuples, what is the optimal assignment of regions of the join matrix to reducers? The optimization function seeks to balance the reducer input (i.e., creating enough k_i keys to balance the load to all reducers) with the cost to produce the output (i.e., not overwhelm any single reduce task with computing a massive Cartesian product of the inputs). The 1-Bucket-Theta algorithm assigns regions of the join matrix to reducers in a randomized fashion to produce a mapping that minimizes job completion time. For each incoming R record, the map function in the 1-Bucket-Theta algorithm randomly chooses a row of the join matrix and emits an output tuple (k_i, v) for each region of S that intersects with this row. Randomization is an important component of the solution because there is no consensus mechanism in MapReduce to find out how many k_i values in the dataset are smaller than the key that the map function is currently processing.

Optimizing MapReduce for Join Processing

The performance of each join algorithm depends on a number of factors, including properties of the input (such as input cardinality, join selectivity, data placement) and the hardware (such as disk throughput, memory capacity). Because of this complex parameter space, many efforts have focused on how to optimize MapReduce for join processing.

The Starfish Project (Herodotou and Babu 2011) develops cost-based optimization methods for Hadoop and the Hadoop Distributed File System. Starfish introduces a profiling module that collects statistical information from MapReduce jobs. These statistics are then forwarded to the “what-if” module that aims to minimize the objective cost function using a cost model. Starfish identifies the split size, the number of reduce tasks, the amount of memory that can

buffer output, and the settings of the multi-pass external sort that computes the inputs to each reduce function as the most crucial performance components of the cost model.

Another avenue for performance improvements are mechanisms to control data placement in MapReduce so as to allow the co-location of related data on the same set of nodes. CoHadoop modifies the data placement policy of the Hadoop distributed file system with a new *locator* property (Eltabakh et al. 2011). Files with the same locator attribute are placed on the same set of nodes, whereas files with no locator are placed using the default locality-oblivious strategy. Co-location can improve I/O performance if datasets are accessed in tandem, but it has important implications for fault tolerance. The probability of losing data because of a failure with co-location is significantly smaller than without co-location, because co-location does not scatter data blocks randomly across the cluster. However, if a failure occurs, more data would be unavailable with co-location. Therefore, co-location decreases the possibility of data loss but amplifies its magnitude.

Database systems optimize a multi-way join by enumerating possible orders of binary joins that construct left-deep, bushy and right-deep plans (Liu and Blanas 2015). The source-agnostic MapReduce framework presents an opportunity to evaluate multi-way joins in a more efficient manner than as a sequence of binary joins. Afrati and Ullman (2010) introduce an algorithm for multi-way join evaluation in MapReduce that is shown to have lower communication cost than evaluating the same multi-way join as a sequence of binary joins. In this algorithm, each join attribute is assigned to a number of “shares.” A “share” corresponds to a hash bucket into which the value of the attribute will be hashed to be transmitted to the appropriate reduce task. The total number of reduce tasks is the product of the “shares” of all attributes. The algorithm uses Lagrange multipliers to optimize the number of “shares” that are assigned to each attribute such that the overall communication cost is minimized.

Other Research Efforts

Ongoing research aims to bridge the gap between the usability of the MapReduce framework and the functionality of database management systems. One research trajectory aims to present a common query interface to the user but synergistically process data between a MapReduce-like framework and a database system in the backend (Bajda-Pawlikowski et al. 2011). Exemplars include HadoopDB (Abouzeid et al. 2009) and Polybase (DeWitt et al. 2013).

The inherent difficulty in expressing arbitrary parallel computation as map and reduce functions motivated the development of big data analytics frameworks for general parallel computation. AsterixDB (2017), Drill (2017), Hive (2017), Impala (2017), Quickstep (2017), and Spark (2017) are all open-source Apache projects that move beyond the MapReduce paradigm for big data processing and natively support parallel joins.

Many machine learning and data mining algorithms are iterative or are designed for datasets that evolve over time. Incremental computation is poorly supported in MapReduce, however, as a task cannot transparently reuse the processing done in a previous phase. Researchers have proposed several solutions for applications that perform incremental computations with new data. Incoop allows unmodified MapReduce programs to reuse intermediate results from previous runs using memoization and stable partitioning (Bhatotia et al. 2011). The resilient distributed dataset abstraction in Spark stores intermediate results in memory to avoid disk I/O across iterations (Zaharia et al. 2012). Dedicated stream processing engines, such as Apache Flink (Carbone et al. 2015), support continuous processing with low latency.

An appealing property of MapReduce is its ability to directly process text data without a cumbersome cleaning and loading step. Research efforts have investigated how database systems can ingest data more efficiently. Invisible loading incrementally imports the data into a database when parsing data in MapReduce (Abouzied et al. 2013). NoDB bypasses data loading in PostgreSQL and directly processes external files

(Alagiannis et al. 2012). Queries in NoDB are accelerated through positional maps and selective parsing. The ScanRAW operator queries data and simultaneously loads frequent accessed data in a database (Cheng and Rusu 2015). This principle can be applied to binary formats such as the HDF5 format for scientific data processing (Blanas et al. 2014).

MapReduce frameworks have adopted storage optimizations found in database systems to accelerate data analytics. Floratou et al. (2011) incorporate column-oriented storage techniques in the Hadoop implementation of MapReduce. Many big data storage formats including RCFFile (He et al. 2011), Apache Parquet (Parquet 2017), and Apache Avro (2017) have embraced columnar data layouts.

References

- Abouzeid A, Bajda-Pawlikowski K, Abadi DJ, Rasin A, Silberschatz A (2009) HadoopDB: an architectural hybrid of mapreduce and DBMS technologies for analytical workloads. *VLDB* 2(1):922–933. <http://www.vldb.org/pvldb/2/vldb09-861.pdf>
- Abouzied A, Abadi DJ, Silberschatz A (2013) Invisible loading: access-driven data transfer from raw files into database systems. In: EDBT
- Afrati FN, Ullman JD (2010) Optimizing joins in a map-reduce environment. In: Proceedings of the 13th international conference on extending database technology, EDBT '10. ACM, New York, pp 99–110. <http://doi.acm.org/10.1145/1739041.1739056>
- Alagiannis I, Borovica R, Branco M, Idreos S, Ailamaki A (2012) NoDB: efficient query execution on raw data files. In: SIGMOD
- AsterixDB (2017) Apache AsterixDB. <https://asterixdb.apache.org/>. Accessed Dec 2017
- Avro (2017) Apache Avro. <https://avro.apache.org/>. Accessed Dec 2017
- Bajda-Pawlikowski K, Abadi DJ, Silberschatz A, Paulson E (2011) Efficient processing of data warehousing queries in a split execution environment. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data, SIGMOD '11. ACM, New York, pp 1165–1176. <http://doi.acm.org/10.1145/1989323.1989447>
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquin R (2011) Incoop: MapReduce for incremental computations. In: Proceedings of the 2nd ACM symposium on cloud computing, SOCC '11. ACM, New York, pp 7:1–7:14. <http://doi.acm.org/10.1145/2038916.2038923>

- Blanas S, Patel JM, Ercegovac V, Rao J, Shekita EJ, Tian Y (2010) A comparison of join algorithms for log processing in MapReduce. In: ACM SIGMOD. <http://doi.acm.org/10.1145/1807167.1807273>
- Blanas S, Wu K, Byna S, Dong B, Shoshani A (2014) Parallel data analysis directly on scientific file formats. In: SIGMOD
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38. <http://sites.computer.org/debull/A15dec/p28.pdf>
- Cheng Y, Rusu F (2015) SCANRAW: a database meta-operator for parallel in-situ processing and loading. TODS 40(3)
- Dean J, Ghemawat S (2004) Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on symposium on operating systems design & implementation – volume 6, OSDI'04. USENIX Association, Berkeley, pp 10–10. <http://dl.acm.org/citation.cfm?id=1251254.1251264>
- DeWitt DJ, Halverson A, Nehme R, Shankar S, Aguilar-Saborit J, Avanes A, Flasza M, Gramling J (2013) Split query processing in Polybase. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD '13. ACM, New York, pp 1255–1266. <http://doi.acm.org/10.1145/2463676.2463709>
- Drill (2017) Apache Drill. <https://drill.apache.org/>. Accessed Dec 2017
- Eltabakh MY, Tian Y, Özcan F, Gemulla R, Krettek A, McPherson J (2011) CoHadoop: flexible data placement and its exploitation in Hadoop. PVLDB 4(9):575–585. <http://www.vldb.org/pvldb/vol4/p575-eltabakh.pdf>
- Floratou A, Patel JM, Shekita EJ, Tata S (2011) Column-oriented storage techniques for mapreduce. PVLDB 4(7):419–429. <http://www.vldb.org/pvldb/vol4/p419-floratou.pdf>
- Hadoop (2017) Apache Hadoop. <https://hadoop.apache.org/>. Accessed Dec 2017
- He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z (2011) RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: Proceedings of the 27th international conference on data engineering, ICDE 2011, 11–16 Apr 2011, Hannover, pp 1199–1208. <https://doi.org/10.1109/ICDE.2011.5767933>
- Herodotou H, Babu S (2011) Profiling, what-if analysis, and cost-based optimization of mapreduce programs. PVLDB 4(11):1111–1122. <http://www.vldb.org/pvldb/vol4/p1111-herodotou.pdf>
- Hive (2017) Apache Hive. <https://hive.apache.org/>. Accessed Dec 2017
- Impala (2017) Apache Impala. <https://impala.apache.org/>. Accessed Dec 2017
- Liu F, Blanas S (2015) Forecasting the cost of processing multi-join queries via hashing for main-memory databases. In: Proceedings of the sixth ACM symposium on cloud computing, SoCC 2015, Kohala Coast, 27–29 Aug 2015, pp 153–166. <http://doi.acm.org/10.1145/2806777.2806944>
- Okcan A, Riedewald M (2011) Processing theta-joins using mapreduce. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data, SIGMOD '11. ACM, New York, pp 949–960. <http://doi.acm.org/10.1145/1989323.1989423>
- Parquet (2017) Apache Parquet. <https://parquet.apache.org>. Accessed Dec 2017
- Quickstep (2017) Apache Quickstep. <https://quickstep.incubator.apache.org>. Accessed Dec 2017
- Spark (2017) Apache Spark. <https://spark.apache.org/>. Accessed Dec 2017
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation, NSDI'12. USENIX Association, Berkeley, pp 2–12. <http://dl.acm.org/citation.cfm?id=2228298.2228301>

Parallel Processing with Big Data

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

P

Synonyms

Big Data Supercomputing; Computational needs of big data

Definitions

Discrepancy between the explosive growth rate in data volumes and the improvement trends in processing and memory access speeds necessitates that parallel processing be applied to the handling of extremely large data sets.

Overview

Both data volumes and processing speeds have been on exponentially rising trajectories since the onset of the digital age (Denning and Lewis 2016), but the former has risen at a

much higher rate than the latter. It follows that parallel processing is needed to bridge the gap. In addition to providing a higher processing capability to deal with the requirements of large data sets, parallel processing has the potential of easing the “von Neumann bottleneck” (Markgraf 2007), sometimes referred to as “the memory wall” because of its tendency to hinder the smooth progress of a computation, when operands cannot be supplied to the processor at the required rate (McKee 2004; Wulf and McKee 1995). Parallel processing algorithms and architectures (Parhami 1999) have been studied since the 1950s as a way of improving computer system performance and, more recently, as a way of continuing the exponential rise in performance while keeping the power consumption in check (Gautschi 2017; Gepner and Kowalik 2006; Koomey et al. 2011).

Trends in Parallel Processing

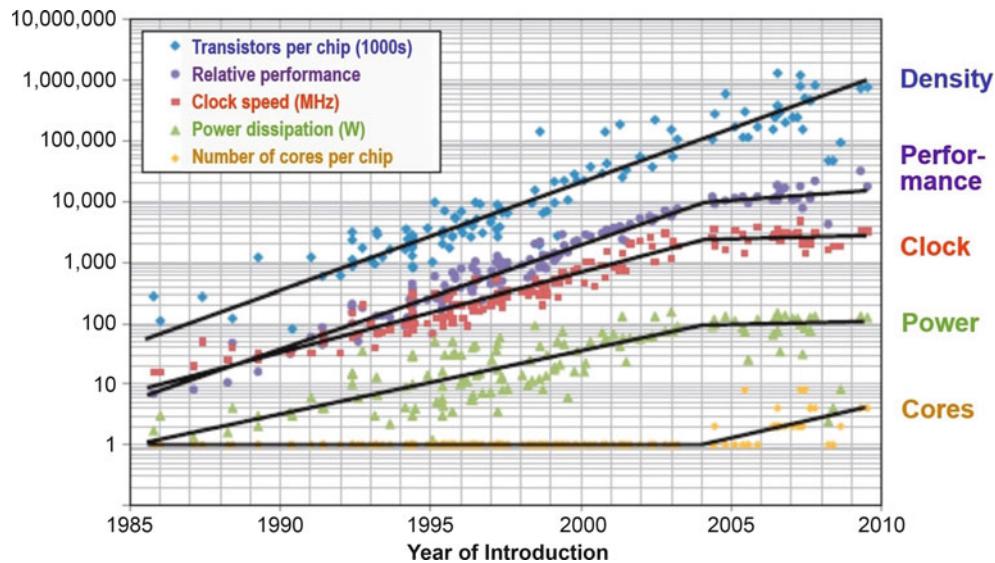
Interest in parallel processing began in the 1960s with the design of ILLIAC IV, generally recognized as the world’s first supercomputer (Hord 2013). The 64-processor machine, later built and operated by Burroughs Corporation, had a single-instruction-stream, multiple-data-stream architecture, SIMD for short (Flynn and Rudd 1996), which uses a single instruction execution unit, with each instruction applied to multiple data items simultaneously. The other main class of parallel architectures is known as MIMD, in which there are multiple instruction streams in addition to multiple data streams. The latter architectural category has a great deal of variation in terms of how memory is implemented (global or distributed) and how the multiple processors communicate and synchronize with each other (shared-variable or message-passing).

Modern supercomputers, including most entries in the Top500 Supercomputers List, which is updated twice a year (Top-500 2017), tend to be MIMD DMMP, meaning that they use distributed memory, whereby processors are endowed with chunks of the memory space, and communicate

via message-passing. Each processor has direct (and thus fast) access to its local memory and indirect (and thus slower) access to nonlocal or remote chunks of memory. Top-of-the-line supercomputers are often used to perform numerically intensive computations involving floating-point numbers. Thus, their performance is measured in floating-point operations per second, or FLOPS. As of late 2017, the top supercomputer in the world had a peak performance of 125 PFLOPS (Peta-FLOPS) and a sustained performance at 75% of the peak. We are now on the verge of achieving a peak performance of 1 EFLOPS (Exa-FLOPS or 10^{18} FLOPS).

The topmost trend line in Fig. 1 shows the exponential rise in the number of transistors that can be placed on a single microchip. This exponential rise in density is known as Moore’s Law, after Intel co-founder Gordon E. Moore, who first observed the trend (Brock and Moore 2006; Mack 2011; Schaller 1997). Up to the early 2000s, the rising chip density was accompanied by exponential improvement in performance, owing to the correspondingly higher clock frequencies. Then, clock frequency and performance trend lines began to flatten out, mostly because the attendant growth in power dissipation led to difficulties in cooling the superdense circuits. Thus, focus on performance enhancement shifted to using multiple independent processors on the same chip, giving rise to multicore processors (Gepner and Kowalik 2006). Using c cores, each with $1/c$, the performance is more energy-efficient, because power consumption is a superlinear function of the single-core performance.

Big data necessitates a reassessment of how we gauge supercomputer performance. While FLOPS rating and IOPS, its integer counterpart, are still important, input/output and storage bandwidth assume a larger role in determining performance. A supercomputer doing scientific calculations may receive a handful of input parameters and a set of equations that define a scientific or engineering model and then compute for days or weeks, before spewing the answers. Many big-data applications require a steady stream of new data to be input, stored, processed, and output, thus possibly straining the memory



Parallel Processing with Big Data, Fig. 1 Trends in transistor density, processor performance, clock frequency, power dissipation, and number of cores on a chip. (NRC 2011)

and I/O bandwidths, which tend to be more limited than computation rate.

Parallel Processing Models

Parallel processing architectures vary greatly in their organizations and user interfaces. Over the years, many abstract models of parallel processing have been introduced in an effort to accurately represent processing, storage, and communication resources, along with their interactions, allowing developers of parallel applications to visualize and take advantage of the available trade-offs, without being bogged down in machine-specific details. At the coarse level, we can distinguish data-parallel and control-parallel approaches to parallel processing (Parhami 1999).

Data parallelism entails partitioning a large data set among multiple processing nodes, with each one operating on an assigned chunk of data, before participating in the process of combining the partial results. It is often the case in big-data applications that the same set of operations must be performed on each data chunk, making SIMD processing the most efficient alternative. In prac-

tice, however, synchronization of a large number of processing nodes introduces overheads and inefficiencies that cut into the speed gains. So, it might be beneficial to direct the processing nodes to execute a single program on multiple data chunks asynchronously, with sparse coordination, giving rise to the SPMD (Darema 2001) parallel processing model.

Google's MapReduce (Dean and Ghemawat 2008, 2010) is the most prominent cloud-based system for SPMD parallel processing. In the map stage, data is distributed within independent tasks on different processing nodes. Each map-stage task produces a set of key-value pairs as its output. These outputs are then fed to reduce-stage tasks, where they are aggregated into a smaller set of key-value pairs that constitute the final output. Hadoop is a widely used open-source implementation of MapReduce on Apache servers (Bu et al. 2010). It consists of Hadoop's version of MapReduce, the Hadoop distributed file system (Shafer et al. 2010; Shvachko et al. 2010), that allows a user to store huge data files across multiple nodes while keeping the image of a centrally located file (stored in one piece), and Hadoop YARN (Vavilapalli et al. 2013), which manages computing resources and allows tasks to be scheduled on them.

The fully open-source Apache Spark (Zaharia et al. 2016) is in many ways similar to MapReduce, with the main differences that it uses a unified engine to replace various other needed subsystems, thus making the programming effort much more manageable and less error-prone, and a data-sharing abstraction, “resilient distributed data sets” or RDDs, which make it significantly more efficient for certain workloads.

In control-parallel schemes, which are more broadly applicable than data parallelism, multiple nodes operate independently on subproblems, synchronizing with each other by sending informational and synchronization messages. The said independence allows heterogeneous and application-specific resources to be employed, without cross-interference or slower resources hindering the progress of faster ones. More on this later. An attractive submodel of control parallelism, that aims to reduce communication and I/O overheads, is bulk synchronous parallel, or BSP, processing (Valiant 1990). BSP computations consist of communication- and synchronization-free supersteps that are executed to conclusion independently, before any communication or synchronization occurs.

Google’s Pregel system (Malewicz et al. 2010) is a practical implementation of BSP for executing iterative graph algorithms, while holding an entire large graph in memory, spread across many nodes, so as to avoid disk access latency. Other examples of the control-parallel paradigm are seen in event processing and stream processing systems commonly used in social media and other notification-driven applications (Abadi et al. 2003; Condie et al. 2010; Eugster et al. 2003; Rixner 2001).

Beyond the two broad kinds of parallel processing, reflected in data-parallel and control-parallel schemes, there are various other kinds of parallelism that can be used as competing or complementary approaches. These include instruction-level parallelism (Rau and Fisher 1993), subword parallelism (Lee 1997), dataflow parallelism (Sakai et al. 1989), and simultaneous-multithreaded parallelism (Eggers et al. 1997), the latter offering the advantage

of lower overhead as well as greater resource sharing in running interdependent tasks in parallel.

Heterogeneity and Hardware Acceleration

Heterogeneous parallel processing entails using multiple processing resources that have diverse designs and performance parameters. Heterogeneity complicates the problems of task assignment and coordination, but it also increases flexibility in matching computational requirements of tasks to hardware and software capabilities. In a heterogeneous system, users can have access to the capabilities of a supercomputer, say, even though their use of such a powerful machine is too limited to justify purchase or even a long-term usage contract.

Within a heterogeneous parallel computing system, some nodes may be specialized for highly efficient execution of certain computations through the use of application-specific hardware-/software and augmentation with accelerators. Examples include use of graphic processing units or GPUs as computation engines (Nvidia 2016; Owens et al. 2008; Singer 2013), units optimized for information retrieval and data mining (Sklyarov et al. 2015), and a variety of other specialized resources made accessible through the cloud (Caulfield et al. 2016).

In addition to fixed acceleration resources, reconfigurable devices, that can be programmed to act as different kinds of acceleration mechanisms, may be deployed. Modern FPGAs (Kuon et al. 2008) offer immense computational resources that can be tailored dynamically to help remove processing bottlenecks. Application-specific circuits, whether custom-designed or implemented on FPGAs, provide additional options for acceleration. For example, sorting networks (Mueller et al. 2012; Parhami 1999) can be used in a wide array of contexts, which include facilitating classification or speeding up subsequent search operations.

Interconnection Networks

Modern parallel computers use commodity processors, often multicore, multithreaded, or GPUs, as computing resources. This allows rapid advances in processor speed and energy efficiency to become immediately available to parallel systems. It follows that the distinction between various parallel processing systems is, to a great extent, determined by the interconnection networks used to allow data exchange between computing nodes. The network connecting processing resources can also be commodity, such as Ethernet. However, it often pays to design a custom network, or at least use a custom configuration of commodity switches, that matches the communication requirements.

Over the years, numerous interconnection network architectures have been proposed and used (Dally and Towles 2004; Duato et al. 2003; Parhami 1999) for different system scales. When computing nodes are on the same electronic chip, they are connected by an on-chip network (Benini and De Micheli 2002), whose design is often more restricted in view of limitations on area and power. Interconnecting processing nodes within a large mainframe or supercomputer system presents challenges in wiring and packaging, with many theoretically superior topologies becoming impractical because they cannot be implemented within the physical limits imposed by partitioning, packaging, signal delays on long wires, and the like (Dally and Towles 2004). Interconnecting servers within a data center presents fewer limitations in terms of topology, but given the scale, energy efficiency considerations, as well as reliability and serviceability factors, become dominant.

Mapping, Scheduling, and Virtualization

A fundamental problem in parallel processing is the mapping of computations to hardware resources. Ideally, this mapping should be automatic and transparent, so as to relieve users from

having to deal with hardware details and configuration changes. However, user-guided mapping can result in better running times and more efficient resource utilization, in view of targeted optimizations in fitting parts of the tasks to capabilities of the available processing resources.

An important kind of mapping, whose roots go back to before computers appeared on the scene, is scheduling. For parallel processing, scheduling refers to the assignment of computational sub-tasks to processing nodes while honoring various constraints, in such a way that an objective function is optimized (Sinnen 2007). The simplest objective function is task completion time, but a variety of more detailed objectives, such as meeting deadlines and minimizing energy consumption, may also enter into the picture.

Virtualization (Nanda and Chiueh 2005) allows system resources to be used effortlessly among different users while isolating and protecting them from one another to ensure privacy and security. The techniques used represent extensions of, and improvements upon, time-sharing schemes of the 1960s (Wilkes 1972) that fell out of favor when compact and inexpensive hardware became available in the form of minicomputers and, eventually, microcomputers. Virtual-machine monitors (Rosenblum and Garfinkel 2005) isolate users from hardware details, resource fluctuations, and configuration changes, thus allowing greater focus on application correctness and high-level trade-offs. Reliability is also improved, both because software bugs are isolated within virtual machines, preventing their spread, and because any detected hardware malfunction can be circumvented by rescheduling affected tasks on other operational virtual machines.

P

Future Directions

While parallel processing technologies have matured over more than five decades, requirements of big-data applications are already creating new challenges, which will pose greater difficulties with the continued exponential growth in data

volumes. This explains the proliferation of proposals for hardware architectures, software aids, and virtualization schemes to ease the key bottlenecks.

Studies on expected direction of computer architecture research over the coming decades (Ceze et al. 2016; Stanford University 2012) point to substantial growth in the use of parallel processing for performance, scalability, energy economy, and reliability reasons. Changes in parallel architectures and attendant virtualization techniques will both influence and be driven by the evolution of cloud computing systems (Agrawal et al. 2011). As data volumes grow and the scope of hardware resources for big-data processing expands, energy efficiency, which is already a major design and cost consideration, will grow in importance (Koomey et al. 2011).

Cross-References

- ▶ [Big Data and Exascale Computing](#)
- ▶ [Computer Architecture for Big Data](#)
- ▶ [Energy Implications of Big Data](#)
- ▶ [GPU-Based Hardware Platforms](#)

References

- Abadi DJ, Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. *Int J Very Large Data Bases* 12(2):120–139
- Agrawal D, Das S, El Abbadi A (2011) Big data and cloud computing: current state and future opportunities. In: Proceedings of 14th international conference on extending database technology, Uppsala, pp 530–533
- Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. *IEEE Comput* 35(1):70–78
- Brock DC, Moore GE (eds) (2006) Understanding Moore's law: four decades of innovation. Chemical Heritage Foundation, Philadelphia
- Bu Y, Howe B, Balazinska M, Ernst MD (2010) HaLoop: efficient iterative data processing on large clusters. *Proc VLDB Endowment* 3(1–2):285–296
- Caulfield AM et al (2016) A cloud-scale acceleration architecture. In: Proceedings of 49th IEEE/ACM international symposium microarchitecture, Orlando, pp 1–13
- Ceze L, Hill MD, Wenisch TE (2016) Arch2030: a vision of computer architecture research over the next 15 years, Computing Community Consortium. Online document. <http://cra.org/ccc/wp-content/uploads/sites/2/2016/12/15447-CCC-ARCH-2030-report-v3-1-1.pdf>
- Condie T, Conway N, Alvaro P, Hellerstein JM, Elmeleegy K, Sears R (2010) MapReduce online. *Proc USENIX Symp Networked Syst Des Implement* 10(4):20
- Dally WJ, Towles BP (2004) Principles and practices of interconnection networks. Elsevier, Amsterdam
- Darema F (2001) The SPMD model: past, present and future. In: Proceedings of European parallel virtual machine/message passing interface users' group meeting, Springer
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Dean J, Ghemawat S (2010) MapReduce: a flexible data processing tool. *Commun ACM* 53(1):72–77
- Denning PJ, Lewis TG (2016) Exponential laws of computing growth. *Commun ACM* 60(1):54–65
- Duato J, Yalamanchili S, Ni LM (2003) Interconnection networks: an engineering approach. Morgan Kaufmann, San Francisco
- Eggers SJ, Emer JS, Levy HM, Lo JL, Stamm RL, Tullsen DM (1997) Simultaneous multithreading: a platform for next-generation processors. *IEEE Micro* 17(5):12–19
- Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. *ACM Comput Surv* 35(2):114–131
- Flynn MJ, Rudd KW (1996) Parallel architectures. *ACM Comput Surv* 28(1):67–70
- Gautschi M (2017) Design of energy-efficient processing elements for near-threshold parallel computing. Doctoral thesis, ETH Zurich
- Gepner P, Kowalik MF (2006) Multi-core processors: new way to achieve high system performance. In: Proceedings of IEEE international symposium on parallel computing in electrical engineering, Bialystok, pp 9–13
- Hord RM (2013) The Illiac IV: the first supercomputer. Springer, Berlin
- Koomey JG, Berard S, Sanchez M, Wong H (2011) Implications of historical trends in the electrical efficiency of computing. *IEEE Ann Hist Comput* 33(3):46–54
- Kuon I, Tessier R, Rose J (2008) FPGA architecture: survey and challenges. *Found Trends Electron Des Autom* 2(2):135–253
- Lee RB (1997) Multimedia extensions for general-purpose processors. In: Proceedings of IEEE workshop signal processing systems, design and implementation, Leicester, pp 9–23
- Mack CA (2011) Fifty years of Moore's law. *IEEE Trans Semicond Manuf* 24(2):202–207
- Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of ACM

- SIGMOD international conference on management of data, Indianapolis, pp 135–146
- Markgraf JD (2007) The von Neumann bottleneck. Online source that is no longer accessible (will find a replacement for this reference during revisions)
- McKee SA (2004) Reflections on the memory wall. In: Proceedings of the conference on computing frontiers, Ischia, pp 162–167
- Mueller R, Teubner J, Alonso G (2012) Sorting networks on FPGAs. Int J Very Large Data Bases 21(1): 1–23
- Nanda S, Chiueh TC (2005) A survey on virtualization technologies, technical report TR179, Department of Computer Science, SUNY at Stony Brook
- NRC (2011) The future of computing performance: game over or next level? Report of the US National Research Council, National Academies Press
- Nvidia (2016) Nvidia Tesla P100: infinite compute power for the modern data center – technical overview. <http://images.nvidia.com/content/tesla/pdf/nvidia-teslap100-techoverview.pdf>. Accessed 14 Dec 2017
- Owens JD et al (2008) GPU computing. Proc IEEE 96(5):879–899
- Parhami B (1999) Chapter 7: Sorting networks. In: Introduction to parallel processing: algorithms and architectures. Plenum Press, New York, pp 129–147
- Rau BR, Fisher JA (1993) Instruction-level parallel processing: history, overview, and perspective. J Supercomput 7(1–2):9–50
- Rixner S (2001) Stream processor architecture. Kluwer, Boston
- Rosenblum M, Garfinkel T (2005) Virtual machine monitors: current technology and future trends. IEEE Comput 38(5):39–47
- Sakai S, Hiraki K, Kodama Y, Yuba T (1989) An architecture of a dataflow single chip processor. ACM SIGARCH Comput Archit News 17(3):46–53
- Schaller RR (1997) Moore's law: past, present and future. IEEE Spectr 34(6):52–59
- Shafer J, Rixner S, Cox AL (2010) The Hadoop distributed filesystem: balancing portability and performance. In: Proceedings of IEEE international symposium on performance analysis of systems & software, White Plains, pp 122–133
- Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: Proceedings of 26th symposium on mass storage systems and technologies, Incline Village, pp 1–10
- Singer G (2013) The history of the modern graphics processor, TechSpot on-line article. <http://www.techspot.com/article/650-history-of-the-gpu/>. Accessed 14 Dec 2017
- Sininen O (2007) Task scheduling for parallel systems. Wiley, Hoboken
- Sklyarov V et al (2015) Hardware accelerators for information retrieval and data mining. In: Proceedings of IEEE conference on information and communication technology research, Bali, pp 202–205
- Stanford University (2012) 21st century computer architecture: a community white paper. http://csl.stanford.edu/~christos/publications/2012.21_stcenturyarchitecture.whitepaper.pdf
- Top-500 Organization (2017) November 2017 list of the world's top 500 supercomputers. <http://www.top500.org/lists/2017/11/>
- Valiant LG (1990) A bridging model for parallel computation. Commun ACM 33(8):103–111
- Vavilapalli VK et al (2013) Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of fourth symposium on cloud computing, Santa Clara, p 5
- Wilkes MV (1972) Time-sharing computer systems. Elsevier, New York
- Wulf W, McKee S (1995) Hitting the wall: implications of the obvious. ACM Comput Archit News 23(1):20–24
- Zaharia M et al (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65

Parallel Spatial Joins

► Query Processing: Joins

Path Queries

► Graph Path Navigation

P

Pattern Recognition

Alessandro Margara

Politecnico di Milano, Milano, Italy

Synonyms

Complex event processing; Event pattern recognition

Definitions

Complex event recognition (CER) refers to the detection of situations of interest – *complex events* – from streams of *primitive events*. Event recognition is performed on the fly, as new events are observed from the input streams, to enable timely reactions.

Overview

Complex event recognition (CER) aims to detect high-level situations of interest, or *complex events*, on the fly from the observation of streams of lower-level *primitive events*, thus offering the opportunity to implement proper reactive or proactive measures (Artikis et al. 2017). Examples of CER come from many different applicative domains. Environmental monitoring applications observe data coming from sensors to detect critical situations and anomalies. Financial applications require constant analysis of stocks to detect trends. Fraud detection tools monitor streams of credit card transactions to prevent frauds.

The research and development of CER focus on two main aspects: (i) identifying suitable *languages* to define complex events from event *patterns* that predicate on the occurrence, content, and temporal relations of primitive events in the input streams and (ii) defining recognition algorithms and processing techniques that efficiently analyze large volumes of input data – *high throughput* – and produce new results as soon as possible, *low delay*.

To concretely illustrate a typical CER pattern, let us refer to the following event definition D expressed in the TESLA language (Cugola and Margara 2010) adopted by the T-Rex event recognition engine (Cugola and Margara 2012a).

Definition D

```
define Fire(area: string)
from Smoke(area=$a) and last
Temp(value>40,area=$a)
within 5 min
from Smoke and
not Rain(area=$a)
between Temp and Smoke
where Fire.area=Smoke.area
```

Intuitively, definition D recognizes a (complex event of type) fire in a given area a whenever (primitive events of) smoke and high temperature in area a are observed from the input streams within 5 min from each other and without any observation of rain in between.

Several CER languages and systems have been proposed in the last years (Etzion and Niblett 2010; Cugola and Margara 2012c; Artikis et al.

2012), which provide different data and temporal models, pattern languages, processing algorithms, and architectures. The remainder of this chapter overviews the dominant approaches to pattern definition and detection.

Applications

As anticipated in the overview, CER is relevant for disparate application domains. Before discussing the CER models and technologies, we present some of these applications.

CER has been adopted for monitoring and surveillance of vehicles. In maritime surveillance, CER can extract trajectories from streams of positions and detect suspicious or potentially dangerous situations, such as loitering, vessels passing through protected areas, and unsafe shipping (Patroumpas et al. 2015). In the context of vehicular ad hoc networks (VANET), CER can detect traffic jams (Terroso-Saenz et al. 2012). In vehicle sharing applications, CER can help matching requests and availability of suitable vehicles (Alli et al. 2012).

CER has also been used to provide real-time access to monitoring data and enable anomaly detection and resource optimization in the context of grid infrastructures (Balis et al. 2011), health applications and hospitals (Yao et al. 2011), and inventory management (Wu et al. 2006).

In online monitoring applications, CER can track the behavior of users on the Web to improve the usability of Web sites or to better select the advertisements (Ali et al. 2009).

Finally, CER can be used to extract statistics and highlights in sports events (Mutschler et al. 2013).

Data Models

Event recognition evaluates input streams of (primitive) events to produce output streams of (complex) events. This section analyzes the most common approaches to model events and event streams.

Events are the core elements of the data model. Each event represents the notification of occurrence of an aspect of interest in the environment under analysis, such as a reading from a sensor. Several formats are possible to encode events,

the most common being attribute/value pairs. Other formats include JSON, XML, or native structures or objects in a specific programming language. Most often, events are characterized by a *type* (or *schema*) that determines their structures. For instance, if an event is encoded using attribute/value pairs, the type determines the number, names, and types of the attributes that appear in the event.

As an example, definition D above defines an event of type *Fire* with a single attribute *area* of type *string*. Within the pattern, it selects events of type *Smoke*, *Temp*, and *Rain*, and predicates on the content of their attributes (*area* and *value*).

Streams can be *homogeneous* or *heterogeneous*. The former include events of the same type, whereas the latter can include events of different types. In the presence of homogeneous streams, the processing engine can be optimized to leverage the fixed structure of the events coming from that stream.

Given the central role of time in event recognition, events always include a temporal attribute or *timestamp*. Depending on the semantics of the recognition, the timestamp can represent the actual time of occurrence of the event, as measured by the event producer (for instance, a sensor), the time in which the event is received by the recognition system, or the time in which the event is processed. In the first case, we say that the recognition adopts *event time* semantics; in the second case it adopts, *ingestion time* semantics; and in the third case, it adopts *processing time* semantics. These policies have various implications on the order in which the events are processed: the interested reader can refer to the “Time Management” chapter.

Furthermore, timestamps can represent a single point in time or an interval. In the second case, they encode the interval of validity of the event. Interval time can be more expressive but complicates the definition of the semantics of event recognition (White et al. 2007).

Event Recognition Languages

Event recognition languages build on a set of primitive operators that predicate on the content

and temporal relations between events and provide abstractions to combine these operators into complex event definitions.

This section first presents the set of operators most commonly found in existing languages and then introduces the main classes of abstractions for event recognition.

Operators for Event Recognition

To present the most common operators for event recognition, let us refer again to definition D. At a high level, it consists of three parts: the *define* clause specifies the type and content of the complex event to be generated upon recognition, the *from* clause specifies the pattern to be recognized, and the *where* clause defines the content – values for the attributes – of the complex event based on the content of primitive events.

Selection. Selection operators isolate primitive events relevant for the pattern based on their type and on the value of their attributes. For instance, definition D *selects* events of type *Smoke* and *Temp* and further constrains the content of *Temp* by requesting a *value* larger than 40. Depending on the specific language, the predicates on the content of events can have different forms, including regular expressions, as well as conjunctions and disjunctions of constraints on one or more attributes.

Conjunctions and sequences. Conjunctions demand for the presence of multiple events by combining individual selection operators. For instance, the pattern in definition D requires the presence of *both* an event of type *Smoke* and an event of type *Temp*. Sequence operators are a special kind of conjunction that further require the events to appear in a specific temporal order. Many languages only offer conjunctions in the form of sequences, thus enforcing the definition of a specific order between the events to be detected.

Temporal constraints. Temporal constraints express time boundaries for the entire pattern or for some parts of it. For instance, Rule R requires

`Smoke` and `Temp` to occur within no more than 5 min from each other.

Parameters. Parameters bind the values of different events that appear in the pattern. For instance, definition D requires that `Smoke` and `Temp` share the same value for the attribute `area`.

Negations. Negations express things that must *not* occur. For instance, the pattern in definition D is recognized if no `Rain` event is detected between the occurrence of `Temp` and `Smoke`. Negations can be only evaluated within temporal boundaries; otherwise it becomes impossible to determine if the negated situation will eventually occur some time in the future. For instance, definition D uses the occurrences of `Temp` and `Smoke` to define the boundaries for the negation.

Aggregates. Aggregates extract a value from the content of multiple primitive events. For instance, definition D could be modified to predicate on the *average* value of the temperature in a given area rather than considering a single occurrence of temperature. Aggregate values can be used both to constrain the validity of a pattern – for instance, by enforcing a minimum threshold to the average temperature measured in a given area – and to define the value of some attribute of the complex event generated.

Hierarchies. Most languages enable reusing complex events within the definition of other complex events, thus promoting modularity and composability. With reference to definition D, a new definition D could reuse the concept of fire in area a to recognize a severe alert when some people are moving toward area a.

Selection and consumption policies. Selection policies define the semantics of recognition when the pattern is satisfied by multiple set of events from the input streams. For instance, definition D specifies that in the presence of multiple temperature events with the desired values and within the given time span only the most recent – the last – has to be considered. Other valid pos-

sibilities would be selecting the oldest – the first – temperature event or selecting all temperature events and generate a fire recognition for each of them.

Consumption policies define which events remain valid after a recognition and which ones do not – they are *consumed*. With reference to definition D, the consumption policy defines what happens in case one temperature event is followed by two smoke events: if the temperature is not consumed after the first recognition, then the second smoke event can reuse the temperature event to trigger a new recognition, otherwise it cannot. By default, TESLA definitions do not consume events, but an optional consuming clause can express the events to be consumed on a per-rule basis.

Selection and consumption policies have been first introduced in the domain of event processing in active database systems (Chakravarthy and Mishra 1994), when they were combined in a set of predefined default choices called *contexts*.

Models for Event Recognition

We now overview different models for event recognition that integrate the operators discussed above (Artikis et al. 2017).

Automata-based models. Automata-based models compile complex event definitions into some form of automata that are later adopted for recognition. Automata states represent primitive events to be recognized: upon recognition of a primitive event, an automaton transitions through an outgoing edge to a subsequent state. Edges can be decorated with pattern constraints that must be satisfied to enable the transition: examples are temporal constraints and parameters.

New automata are instantiated on demand to initiate a new recognition: an automaton reaches an accepting state and produces a complex event when it has recognized all the primitive events in the pattern; conversely, an automaton reaches a failure state and gets deleted if it violates some constraint expressed in the pattern.

Examples of languages and systems that adopt automata-based models include Cayuga (Brenna

et al. 2007), SASE (Wu et al. 2006), and SASE+ (Gyllstrom et al. 2008).

Languages that rely on this model resemble regular expressions that concatenate individual events into sequences. In some cases, regular expressions predicate over contiguous events on the input streams (Brenna et al. 2007), while in other cases, they accept and ignore the presence of intermediate events within the stream that are not related with the pattern (Wu et al. 2006). Some languages offer operators similar to Kleene plus in regular expressions to define repetitions of given sub-patterns one or more times.

Automata-based models typically assume that the events in a stream are received in order and exploit this property to advance the automata as soon as an event becomes available, without waiting for possible events received out of order.

Tree-based models. Tree-based models combine operators hierarchically to form a tree. When the operator representing a node of the tree is satisfied, the parent node is evaluated. A complex event is generated when the operator in the root node is satisfied.

A tree-based model promotes sharing of operators across definitions and decouples the order of arrival of events in the stream from the order of evaluation of operators.

Examples of systems that adopt tree-based models include ZStream (Mei and Madden 2009), which uses operator trees for pattern specification, and NextCEP (Schultz-Møller et al. 2009) and Esper (Esper 2017), which adopt tree-based structures for pattern recognition.

Logic-based models. Logic-based models ground the semantics of recognition into some logic-based formalism. In some cases, systems that rely on logic-based models encode definitions using logic programming and use inference to detect complex events. For instance, ETALIS (Anicic et al. 2011) builds on the Prolog language and inference mechanism. In most cases, however, logic-based definitions are translated to exploit ad hoc data structures and algorithms for improved efficiency. For instance, the semantics of TESLA definitions are defined in terms of metric tempo-

ral logic formulas, but recognition adopts custom algorithms.

We present two main logic-based formalisms that are considered good representatives of logic-based modeling (Artikis et al. 2017): chronicle recognition (Dousson and Le Maigat 2007) and event calculus (Artikis et al. 2012).

Chronicle recognition relies on temporal logic and encodes event recognition as logic predicates that evaluate the content and time of occurrence of primitive events, as well as temporal and contextual constraints. As already mentioned, TESLA is a representative of this model.

Event calculus builds on fluents, which are properties holding different values over time. Event calculus specifies how the occurrences of events alter the values of fluents. Situations of interest are encoded as fluents and recognition translates to observing specific values in some fluents. An example of this model is the Real-Time Event Calculus (RTEC) (Artikis et al. 2015).

Event Recognition Algorithms

This section discusses algorithms and techniques to efficiently recognize patterns in streams of events.

P

Incremental algorithms. Automata-based models are often associated to incremental recognition algorithms, which instantiate new automata on demand to represent partial recognition and use the input events to move each automaton from state to state until it reaches an accepting state – and recognizes a complex event – or it reaches a failure state and it is discarded. Incremental algorithms can efficiently recognize sequences of events with content and temporal constraints. One main limitation of these algorithms is that the number of candidate automata can rapidly become very large (Zhang et al. 2014), thus requiring a large amount of memory and forcing the analysis of many partial automata upon the arrival of new events. To mitigate this problem, various compression techniques have been studied that combine common parts shared by different automata (Wu et al. 2006).

Lazy algorithms. Lazy algorithms accumulate events and postpone expensive analysis as much as possible. For instance, tree-based models decouple the order of events from the order of evaluation: each node in the tree stores events in a buffer and produces results upon request. This approach enables optimizing the order of evaluation – query optimization (Schultz-Møller et al. 2009) – and parallelizing the analysis of independent nodes.

Lazy algorithms have been also proposed for automata-based (Kolchinsky et al. 2015) and for logic-based models (Cugola and Margara 2012b), where they proved to be suitable for implementations that take advantage of many core hardware such as GPUs.

Summary

Complex event recognition aims to detect situations of interest – complex events – from streams of primitive events on the fly, to enable timely reactions. The research and development of event recognition target two main aspects: the definition of suitable languages to define complex events from primitive ones and the design of efficient processing algorithms and techniques. Different language models exist that define and combine basic operators using automata, tree-based, or logic models. Recent evaluation algorithms and techniques aim to adopt data structures that are memory efficient and enable parallel processing within and across event definitions.

Cross References

- ▶ [Definition of Data Streams](#)
- ▶ [Stream Processing Languages and Abstractions](#)
- ▶ [Management of Time](#)
- ▶ [Stream Query Optimization](#)

References

- Ali MH, Gerea C, Raman BS, Sezgin B, Tarnavski T, Verona T, Wang P, Zabback P, Ananthanarayanan A, Kirilov A, Lu M, Raizman A, Krishnan R, Schindlauer R, Grabs T, Bjeletich S, Chandramouli B, Goldstein J, Bhat S, Li Y, Di Nicola V, Wang X, Maier D, Grell S, Nano O, Santos I (2009) Microsoft CEP server and online behavioral targeting. VLDB 2(2):1558–1561. <https://doi.org/10.14778/1687553.1687590>
- Alli G, Baresi L, Bianchessi A, Cugola G, Margara A, Morzenti A, Ongini C, Panigatti E, Rossi M, Rotondi S, Savaresi S, Schreiber FA, Sivieri A, Tanca L, Vannutelli Depoli E (2012) Green move: towards next generation sustainable smartphone-based vehicle sharing. In: Proceedings of the conference on sustainable internet and ICT for sustainability (SustainIT'12). IEEE
- Anicic D, Fodor P, Rudolph S, Stühmer R, Stojanovic N, Studer R (2011) ETALIS: rule-based reasoning in event processing. Springer, Berlin/Heidelberg, pp 99–124. https://doi.org/10.1007/978-3-642-19724-6_5
- Artikis A, Skarlatidis A, Portet F, Paliouras G (2012) Review: logic-based event recognition. Knowl Eng Rev 27(4):469–506. <https://doi.org/10.1017/S026988912000264>
- Artikis A, Sergot M, Paliouras G (2015) An event calculus for event recognition. Trans Knowl Data Eng 27(4):895–908
- Artikis A, Margara A, Ugarte M, Vansummeren S, Weidlich M (2017) Complex event recognition languages: tutorial. In: Proceedings of the international conference on distributed and event-based systems (DEBS'17). ACM, pp 7–10. <https://doi.org/10.1145/3093742.3095106>
- Balis B, Kowalewski B, Bubak M (2011) Real-time grid monitoring based on complex event processing. Futur Gener Comput Syst 27(8):1103–1112. <https://doi.org/10.1016/j.future.2011.04.005>
- Brenna L, Demers A, Gehrke J, Hong M, Ossher J, Panda B, Riedewald M, Thatte M, White W (2007) Cayuga: a high-performance event processing engine. In: Proceedings of the international conference on management of data (SIGMOD'07). ACM, pp 1100–1102. <https://doi.org/10.1145/1247480.1247620>
- Chakravarthy S, Mishra D (1994) Snoop: an expressive event specification language for active databases. Data Knowl Eng 14(1):1–26. [https://doi.org/10.1016/0169-023X\(94\)90006-X](https://doi.org/10.1016/0169-023X(94)90006-X)
- Cugola G, Margara A (2010) Tesla: a formally defined event specification language. In: Proceedings of the international conference on distributed event-based systems (DEBS'10). ACM, pp 50–61. <https://doi.org/10.1145/1827418.1827427>
- Cugola G, Margara A (2012a) Complex event processing with t-rex. J Syst Softw 85(8):1709–1728. <https://doi.org/10.1016/j.jss.2012.03.056>
- Cugola G, Margara A (2012b) Low latency complex event processing on parallel hardware. J Parallel Distrib Comput 72(2):205–218. <https://doi.org/10.1016/j.jpdc.2011.11.002>
- Cugola G, Margara A (2012c) Processing flows of information: from data stream to complex event processing. ACM Comput Surv 44(3):15:1–15:62. <https://doi.org/10.1145/2187671.2187677>

- Dousson C, Le Maigat P (2007) Chronicle recognition improvement using temporal focusing and hierarchization. In: Proceedings of the international joint conference on artificial intelligence (IJCAI'07), Morgan Kaufmann, pp 324–329
- Esper (2017) <http://www.espertech.com/esper/>
- Etzion O, Niblett P (2010) Event processing in action. Manning Publications Co., Greenwich
- Gyllstrom D, Agrawal J, Diao Y, Immerman N (2008) On supporting kleene closure over event streams. In: Proceedings of the international conference on data engineering (ICDE'08). IEEE, pp 1391–1393. <https://doi.org/10.1109/ICDE.2008.4497566>
- Kolchinsky I, Sharfman I, Schuster A (2015) Lazy evaluation methods for detecting complex events. In: Proceedings of the international conference on distributed event-based systems (DEBS'15). ACM, pp 34–45. <https://doi.org/10.1145/2675743.2771832>
- Mei Y, Madden S (2009) Zstream: a cost-based query processor for adaptively detecting composite events. In: Proceedings of the international conference on management of data (SIGMOD'09). ACM, pp 193–206. <https://doi.org/10.1145/1559845.1559867>
- Mutschler C, Ziekow H, Jerzak Z (2013) The debs 2013 grand challenge. In: Proceedings of the international conference on distributed event-based systems (DEBS'13). ACM, pp 289–294. <https://doi.org/10.1145/2488222.2488283>
- Patroumpas K, Artikis A, Katzouris N, Vodas M, Theodoridis Y, Pelekis N (2015) Event recognition for maritime surveillance. In: Proceedings of the international conference on extending database technology (EDBT'15), OpenProceedings.org, pp 629–640. <https://doi.org/10.5441/002/edbt.2015.63>
- Schultz-Møller NP, Migliavacca M, Pietzuch P (2009) Distributed complex event processing with query rewriting. In: Proceedings of the international conference on distributed event-based systems (DEBS'09). ACM, pp 4:1–4:12. <https://doi.org/10.1145/1619258.1619264>
- Terroso-Saenz F, Valdes-Vela M, Sotomayor-Martinez C, Toledo-Moreo R, Gomez-Skarmeta AF (2012) A cooperative approach to traffic congestion detection with complex event processing and vanet. Trans Intell Transp Syst 13(2):914–929. <https://doi.org/10.1109/TITS.2012.2186127>
- White W, Riedwald M, Gehrke J, Demers A (2007) What is “next” in event processing? In: Proceedings of the symposium on principles of database systems (PODS'07). ACM, pp 263–272. <https://doi.org/10.1145/1265530.1265567>
- Wu E, Diao Y, Rizvi S (2006) High-performance complex event processing over streams. In: Proceedings of the international conference on management of data (SIGMOD'06). ACM, pp 407–418, <https://doi.org/10.1145/1142473.1142520>
- Yao W, Chu CH, Li Z (2011) Leveraging complex event processing for smart hospitals using RFID. J Netw Comput Appl 34(3):799–810. <https://doi.org/10.1016/j.jnca.2010.04.020>
- Zhang H, Diao Y, Immerman N (2014) On complexity and optimization of expensive queries in complex event processing. In: Proceedings of the international conference on management of data (SIGMOD'14). ACM, pp 217–228. <https://doi.org/10.1145/2588555.2593671>

Pedigree

- Data Provenance for Big Data Security and Accountability

Performance Consortia

- TPC

Performance Evaluation of Big Data Analysis

Jorge Veiga, Roberto R. Expósito, and Juan Touriño
Computer Architecture Group, Universidade da Coruña, A Coruña, Spain

P

Synonyms

- Big Data performance characterization

Definitions

Evaluating the performance of Big Data systems is the usual way of getting information about the expected execution time of analytics applications. These applications are generally used to extract meaningful information from very large input datasets. There exist many high-level frameworks for Big Data analysis, each one oriented to different fields like machine learning and data mining, like Mahout (Apache Mahout 2009), or graph analytics like Giraph (Avery 2011). These high-level frameworks allow to define complex data

processing pipelines that are later decomposed into more fine-grained operations in order to be executed by Big Data processing frameworks like Hadoop (Dean and Ghemawat 2008), Spark (Zaharia et al. 2016), and Flink (Apache Flink 2014). Therefore, the performance evaluation of these frameworks is key to determine their suitability for scalable Big Data analysis.

Big Data processing frameworks can be broken down in several layers, which typically include a data processing engine (e.g., Hadoop MapReduce), a resource manager (e.g., YARN), and a distributed storage system (e.g., HDFS). In order to provide scalability, these frameworks are deployed over the nodes of a cluster, which has a certain set of characteristics (e.g., number of nodes, CPU model, disk, and network technology). Hence, the performance of Big Data systems is affected by multiple factors related both to the software components of the frameworks and the available resources in the cluster.

Most performance evaluation studies are oriented to compare several Big Data frameworks and/or different configuration alternatives. In order to do so, a set of experiments is carried out, which involves generating some input datasets, process them using several representative Big Data workloads and extract the corresponding performance metrics. The obtained results are then analyzed to find performance bottlenecks or potential optimizations.

Overview

The performance evaluation of Big Data systems typically takes into account at least one of the following metrics:

Execution Time

Execution time is generally regarded as the main performance metric. It determines the wall-clock time that users have to wait for the result of their applications when executed with a particular Big Data framework.

Scalability

Scalability is the ability of a Big Data system to increase its performance when adding more resources. Two different kinds of scalability can be considered, vertical scalability (scaling up) and horizontal scalability (scaling out). On the one hand, vertical scalability involves obtaining faster nodes with more powerful processors and more memory. On the other hand, horizontal scalability involves adding more nodes to the system and operates in a distributed environment.

Resource Utilization

The leveraging of system resources (e.g., CPU, network, disk) determines the adaptability of the frameworks to a particular system. Performance bottlenecks can be due to underutilization or overloading of these resources.

Energy Efficiency

Performance is closely related to energy efficiency, as the energy consumed when executing a workload is determined by the power specifications of the system and the execution time. Modifying the underlying system by increasing the cluster size or the computational capabilities of the nodes can reduce execution time, but it may increase the power consumption of the workload. Therefore, determining whether an optimization can improve energy efficiency requires a thorough analysis.

Microarchitectural Behavior

Accessing hardware performance counters that are present in modern processors, like the number of instructions executed or cache misses, can provide meaningful information to characterize the interaction between frameworks and hardware. Moreover, these metrics can be utilized to compare different alternatives in a more fine-grained fashion than just considering overall metrics like execution time. The values of the counters can be accessed in several ways, using APIs like PAPI (Browne et al. 2000) or monitoring tools like perf and Oprofile.

Key Research Findings

Many works have assessed and optimized the performance of Big Data systems taking into account different factors. A summary of the main results obtained is provided next.

Data Processing Engine

A crucial factor for the performance of Big Data frameworks is the underlying data processing engine, which defines the kind of operations that the user can perform to process the input dataset. MapReduce has been one of the most popular batch engines so far, and Hadoop is its de facto standard implementation. However, Hadoop presents some performance overheads, like the writing of intermediate results to disk, which has led to the development of different alternatives that optimize its performance. They whether modify some of its components, like NativeTask (Yang et al. 2013), or redesign the entire underlying architecture, like Flame-MR (Veiga et al. 2016c). Although these works significantly improve the performance of Hadoop, they are still limited by the nature of the MapReduce model.

More advanced in-memory frameworks like Spark and Flink are designed to provide a wider range of data operators than MapReduce, as well as support for other scenarios (e.g., real-time/streaming processing). Although they support map and reduce functions, existing MapReduce applications must be adapted to their new programming model. The increased flexibility of Spark and Flink along with the caching of intermediate results in memory can reduce Hadoop execution times by 77% and 70% on average, respectively (Veiga et al. 2016a).

Other works have explored the possibilities offered by paradigms traditionally oriented to high-performance computing (HPC). For example, parallel programming paradigms like the message passing interface (MPI) can increase significantly the performance of Big Data workloads (Liang et al. 2014; González et al. 2017). However, MPI offers a low-level API that provides poor programming productivity compared to MapReduce, so its use is not feasible for real Big Data scenarios.

File System

Big Data frameworks typically use the Hadoop Distributed File System (HDFS) to distribute the storage of large datasets over the nodes of a cluster, collocating storage and compute services on the same nodes. However, its use is not widespread in HPC systems, which separate compute and storage services by using parallel file systems like GPFS, OrangeFS, or Lustre. This situation has caused the appearance of several works that evaluate the performance of both storage approaches, concluding that GPFS behaves better at low concurrency, while HDFS is more suited to high concurrency (Fadika et al. 2012). Some other works have shown that parallel file systems can also provide performance improvements, like MARIANE (Fadika et al. 2014), which uses a custom MapReduce implementation onto GPFS. Another approach presented by Xuan et al. (2017) uses a two-level storage system by integrating the in-memory file system Tachyon with OrangeFS to obtain higher performance than just using HDFS.

P

Disk Technology

Traditional hard disk drives (HDDs) are progressively being replaced by faster technologies like solid-state drives (SSDs), which obtain significantly better performance but at higher cost per byte. Using SSDs has been reported to improve the performance of Big Data frameworks. Hadoop can store HDFS data and intermediate results on SSDs to eliminate disk bottlenecks when executing I/O-bound workloads. However, SSD disks can increase dramatically the total cost of the system components (Moon et al. 2014). Regarding Spark, its performance can be improved by 23% when using SSDs to store intermediate results compared to the memory-only approach (Choi et al. 2015).

SSDs can also be leveraged actively, performing some simple operations over the stored data. In Lee et al. (2016), a new technique called external sorting utilizes SSDs to perform sort operations during the MapReduce phase, reducing the execution time of Hadoop by up to 36%.

Network Interconnects

Early works claimed that the use of high-performance interconnects like InfiniBand would not have a great impact on the execution time of MapReduce workloads, unless the shuffle algorithm and communication protocol were modified (Fadika et al. 2012). However, more modern evaluations have shown that up-to-date Hadoop versions can leverage this kind of networks by using IP over InfiniBand (IPoIB), obtaining significant performance improvements (Veiga et al. 2016b).

Other works modify the shuffle components to take advantage of remote direct memory access (RDMA) communications. That is the case of the network-levitated merge algorithm (Wang et al. 2011) and RDMA-Hadoop (Wasi-Ur-Rahman et al. 2013). The latter claims to reduce the execution time of Hadoop and the network-levitated merge algorithm by 32% and 21%, respectively, for the TeraSort benchmark. The use of RDMA has also been studied for Spark, showing up to 46% performance improvement for several workloads (Lu et al. 2016b).

Memory Management

As Big Data applications read and generate a lot of data, managing the available memory resources correctly is key to achieve good performance and avoid memory overflows. Most Big Data frameworks are written in some managed language (e.g., Java, Scala) executed by the Java virtual machine (JVM). In this context, objects are tracked to release their memory once they stop being referenced. This process is performed by the garbage collector and can cause significant performance overheads when processing large datasets.

Modifying the original JVM memory management to adapt it to the characteristics of Big Data systems can lead to significant performance improvement. That is shown by proposals like Broom (Gog et al. 2015), which uses a region-based algorithm to allocate data objects. Another example, Yak (Nguyen et al. 2016), implements a hybrid approach that utilizes generation-based and region-based algorithms for control and data objects, respectively. As these memory managers

are implemented in Java, they can be generically applied to any JVM-based framework.

Other solutions are specific to a certain framework, like Deca (Lu et al. 2016a), which modifies the management of data containers in Spark to estimate their lifetime, allocating memory regions accordingly. This mechanism, combined with other optimizations like the use of byte arrays to store data objects, is able to achieve up to 41.6× speedup in cases with data spilling.

Manycore Accelerators

The great majority of Big Data frameworks rely only on CPUs to perform the computations. However, some works propose the use of manycore accelerators, typically available in heterogeneous systems, like GPUs, FPGAs, or Xeon Phi accelerators.

GPUs are a suitable option to accelerate Big Data workloads due to their widespread use and high degree of data parallelism. For example, Mars (Fang et al. 2011) supports the processing of MapReduce workloads using CPU, GPU, or hybrid computations. The combination of Hadoop with Mars can provide a maximum speedup of 2.8. GPUs have also been employed to improve the performance of Spark (Yuan et al. 2016) and Flink (Chen et al. 2017).

Another popular type of accelerator is the Xeon Phi manycore processor that can execute unmodified CPU code. However, the source code of Big Data frameworks must be adapted in order to fully leverage the computational power of this accelerator, as presented by Lu et al. (2015).

Finally, FPGAs are hardware devices that can be programmed to build custom accelerators. Neshatpour et al. (2015) assess the benefits of accelerating typical machine learning and data mining applications by offloading some of their kernels to FPGAs. They obtain a speedup of 2.72, although it must be taken into account that the accelerated framework is highly application-specific.

System Architecture

Some studies have evaluated the performance of “big” Intel Xeon nodes compared to the use of “small” nodes like ARM or Intel Atom. Loghin

et al. (2015) state that “big” nodes are more efficient for CPU-intensive jobs, while “small” ones perform better for I/O-intensive workloads. Malik et al. (2015) conclude that “big” nodes are more efficient as the computational size of the problem increases.

Other evaluations focus on determining whether horizontal scalability is more beneficial than vertical scalability. The results are highly dependent on the framework used and the workload characterization. In general terms, horizontal scalability provides better performance for Hadoop (Li and Shen 2017), while Spark presents better energy efficiency when using vertical scalability (Yoo et al. 2016).

Examples of Application

A wide range of tools have been developed to evaluate the performance of Big Data systems and frameworks. Most of them are benchmark suites that provide multiple kinds of workloads to assess the performance of different use cases. Although the great majority is oriented to Hadoop, like HiBench (Huang et al. 2010) or BigDataBench (Wang et al. 2014), some new ones target in-memory frameworks, like SparkBench (Li et al. 2017) for Spark.

Other evaluation tools not only provide a set of benchmarks but also ease the execution of the experiments. That is the case of MRBS (Sangroya et al. 2012), which is able to automatically set up a Hadoop cluster in a public cloud provider. Once the cluster is running, it injects the dataset and executes the workloads, obtaining execution time, throughput, and cost metrics.

BDEV (formerly MREv, Veiga et al. 2015) is another evaluation tool that supports several flavors of Hadoop, Spark, and Flink. Once the user configures a set of experimental parameters, it launches the frameworks, generates the input datasets, and performs the experiments. It also automatically records several metrics, including execution time, resource utilization, energy efficiency, and hardware performance counters.

Big Data Watchdog (BDWatchdog) (Enes et al. 2017) is another tool that enables to record resource utilization statistics for the individual processes involved in the execution of the frameworks (e.g., DataNode, Spark Executor). Moreover, it can provide real-time profiling information about the execution of the JVM code.

Future Directions for Research

As explained in previous sections, performance is affected by multiple factors. Although different studies have addressed the performance evaluation of Big Data analysis, end users rarely benefit from the research findings provided by these studies. There is still work to be done regarding the development of tools that can bring more meaningful insights to users.

Users that want to select a Big Data framework to utilize in a certain infrastructure can compare their options by performing several experiments. This involves the deployment of several frameworks (e.g., Hadoop, Spark) and testing their performance. In addition to the effort associated to this evaluation, the optimal choice may still require a more thorough analysis, taking also into account different classes of workloads that the user may be willing to execute (e.g., CPU-intensive, I/O-intensive) and varying the configuration parameters of the frameworks (e.g., maps per node, executors per node). Furthermore, some information must be given to the user regarding the performance bottlenecks that may exist and which system resource may be the best option for improving.

Although there are some evaluation tools that ease the execution of these tasks, future ones must incorporate some knowledge to help users to make decisions, not only retrieving raw performance information.

Cross-References

- ▶ [Energy Efficiency in Big Data Analysis](#)

References

- Apache Flink (2014) Scalable batch and stream data processing. <http://flink.apache.org/>, [Last visited: Dec 2017]
- Apache Mahout (2009) Scalable machine learning and data mining. <http://mahout.apache.org/>, [Last visited: Dec 2017]
- Avery C (2011) Giraph: large-scale graph processing infrastructure on Hadoop. In: 2011 Hadoop summit, Santa Clara, pp 5–9
- Browne S, Dongarra J, Garner N, Ho G, Mucci P (2000) A portable programming interface for performance evaluation on modern processors. *Int J High Perform Comput Appl* 14(3):189–204
- Chen C, Li K, Ouyang A, Tang Z, Li K (2017) GPU-accelerated parallel hierarchical extreme learning machine on Flink for Big Data. *IEEE Trans Syst Man Cybern Syst* 47(10):2740–2753
- Choi IS, Yang W, Kee YS (2015) Early experience with optimizing I/O performance using high-performance SSDs for in-memory cluster computing. In: 2015 IEEE international conference on Big Data (IEEE BigData 2015), Santa Clara, pp 1073–1083
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Enes J, Expósito RR, Touriño J (2017) Big Data watchdog: real-time monitoring and profiling. <http://bdwatchdog.dec.udc.es>, [Last visited: Dec 2017]
- Fadika Z, Govindaraju M, Canon R, Ramakrishnan L (2012) Evaluating Hadoop for data-intensive scientific operations. In: 5th IEEE international conference on cloud computing (CLOUD'12), Honolulu, pp 67–74
- Fadika Z, Dede E, Govindaraju M, Ramakrishnan L (2014) MARIANE: using MApReduce in HPC environments. *Futur Gener Comput Syst* 36:379–388
- Fang W, He B, Luo Q, Govindaraju NK (2011) Mars: accelerating MapReduce with graphics processors. *IEEE Trans Parallel Distrib Syst* 22(4):608–620
- Gog I, Giceva J, Schwarzkopf M, Vaswani K, Vytiniotis D, Ramalingam G, Costa M, Murray D, Hand S, Isard M (2015) Broom: sweeping out garbage collection from Big Data systems. In: 15th workshop on hot topics in operating systems (HotOS'15), Kartause Ittingen
- González P, Pardo XC, Penas DR, Teijeiro D, Banga JR, Doallo R (2017) Using the cloud for parameter estimation problems: comparing Spark vs MPI with a case-study. In: 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid 2017), Madrid, pp 797–806
- Huang S, Huang J, Dai J, Xie T, Huang B (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 26th IEEE international conference on data engineering workshops (ICDEW'10), Long Beach, pp 41–51
- Lee YS, Quero LC, Kim SH, Kim JS, Maeng S (2016) ActiveSort: efficient external sorting using active SSDs in the MapReduce framework. *Futur Gener Comput Syst* 65:76–89
- Li Z, Shen H (2017) Measuring scale-up and scale-out Hadoop with remote and local file systems and selecting the best platform. *IEEE Trans Parallel Distrib Syst* 28(11):3201–3214
- Li M, Tan J, Wang Y, Zhang L, Salapura V (2017) SparkBench: a Spark benchmarking suite characterizing large-scale in-memory data analytics. *Clust Comput* 20(3):2575–2589
- Liang F, Feng C, Lu X, Xu Z (2014) Performance benefits of DataMPI: a case study with BigDataBench. In: 4th workshop on Big Data benchmarks, performance optimization and emerging hardware (BPOE'14), Salt Lake City, pp 111–123
- Loghin D, Tudor BM, Zhang H, Ooi BC, Teo YM (2015) A performance study of Big Data on small nodes. *Proc VLDB Endowment* 8(7):762–773
- Lu M, Liang Y, Huynh HP, Ong Z, He B, Goh RSM (2015) MrPhi: an optimized MapReduce framework on Intel Xeon Phi coprocessors. *IEEE Trans Parallel Distrib Syst* 26(11):3066–3078
- Lu L, Shi X, Zhou Y, Zhang X, Jin H, Pei C, He L, Geng Y (2016a) Lifetime-based memory management for distributed data processing systems. *Proc VLDB Endowment* 9(12):936–947
- Lu X, Shankar D, Gugnani S, Panda DK (2016b) High-performance design of Apache Spark with RDMA and its benefits on various workloads. In: 2016 IEEE international conference on Big Data (IEEE BigData 2016), Washington, DC, pp 253–262
- Malik M, Rafatirah S, Sasan A, Homayoun H (2015) System and architecture level characterization of Big Data applications on big and little core server architectures. In: 2015 IEEE international conference on Big Data (IEEE BigData 2015), Santa Clara, pp 85–94
- Moon S, Lee J, Kee YS (2014) Introducing SSDs to the Hadoop MapReduce framework. In: 7th IEEE international conference on cloud computing (CLOUD'14), Anchorage, pp 272–279
- Neshatpour K, Malik M, Ghodrat MA, Sasan A, Homayoun H (2015) Energy-efficient acceleration of Big Data analytics applications using FPGAs. In: 2015 IEEE international conference on Big Data (IEEE BigData 2015), Santa Clara, pp 115–123
- Nguyen K, Fang L, Xu GH, Demsky B, Lu S, Alamian S, Mutlu O (2016) Yak: a high-performance Big-Data-friendly garbage collector. In: 12th USENIX symposium on operating systems design and implementation (OSDI'16), Savannah, pp 349–365
- Sangroya A, Serrano D, Bouchenak S (2012) MRBS: towards dependability benchmarking for Hadoop MapReduce. In: 18th international Euro-par conference on parallel processing workshops (Euro-Par'12), Rhodes Island, pp 3–12
- Veiga J, Expósito RR, Taboada GL, Touriño J (2015) MREv: an automatic MapReduce evaluation tool for Big Data workloads. In: International conference on computational science (ICCS'15), Reykjavík, pp 80–89

- Veiga J, Expósito RR, Pardo XC, Taboada GL, Touriño J (2016a) Performance evaluation of Big Data frameworks for large-scale data analytics. In: 2016 IEEE international conference on Big Data (IEEE BigData 2016), Washington, DC, pp 424–431
- Veiga J, Expósito RR, Taboada GL, Touriño J (2016b) Analysis and evaluation of MapReduce solutions on an HPC cluster. *Comput Electr Eng* 50:200–216
- Veiga J, Expósito RR, Taboada GL, Touriño J (2016c) Flame-MR: an event-driven architecture for MapReduce applications. *Futur Gener Comput Syst* 65:46–56
- Wang Y, Que X, Yu W, Goldenberg D, Sehgal D (2011) Hadoop acceleration through network levitated merge. In: International conference for high performance computing, networking, storage and analysis (SC'11), Seattle, pp 57:1–57:10
- Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y, Gao W, Jia Z, Shi Y, Zhang S, Zheng C, Lu G, Zhan K, Li X, Qiu B (2014) BigDataBench: a Big Data benchmark suite from Internet services. In: 20th IEEE international symposium on high-performance computer architecture (HPCA'14), Orlando, pp 488–499
- Wasi-Ur-Rahman M, Islam NS, Lu X, Jose J, Subramoni H, Wang H, Panda DK (2013) High-performance RDMA-based design of Hadoop MapReduce over InfiniBand. In: 27th IEEE international parallel and distributed processing symposium workshops and PhD forum (IPDPSW'13), Boston, pp 1908–1917
- Xuan P, Ligon WB, Srimani PK, Ge R, Luo F (2017) Accelerating Big Data analytics on HPC clusters using two-level storage. *Parallel Comput* 61:18–34
- Yang D, Zhong X, Yan D, Dai F, Yin X, Lian C, Zhu Z, Jiang W, Wu G (2013) NativeTask: a Hadoop compatible framework for high performance. In: 2013 IEEE international conference on Big Data (IEEE BigData'13), Santa Clara, pp 94–101
- Yoo T, Yim M, Jeong I, Lee Y, Chun ST (2016) Performance evaluation of in-memory computing on scale-up and scale-out cluster. In: 8th international conference on ubiquitous and future networks (ICUFN'16), Vienna, pp 456–461
- Yuan Y, Salmi MF, Huai Y, Wang K, Lee R, Zhang X (2016) Spark-GPU: an accelerated in-memory data processing engine on clusters. In: 2016 IEEE international conference on Big Data (IEEE BigData'16), Washington, DC, pp 273–283
- Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I (2016) Apache Spark: a unified engine for Big Data processing. *Commun ACM* 59(11):56–65

Pipeline Benchmark

► End-to-End Benchmark

Power Benchmarking

► Energy Benchmarking

Predictive Business Process Monitoring

Chiara Di Francescomarino

Fondazione Bruno Kessler – FBK, Trento, Italy

Synonyms

Forecasting business process future; Predictive monitoring of business processes

Definitions

Predictive monitoring of business processes aims at predicting the future of an ongoing (uncomplete) process execution. Predictions related to the future of an ongoing process execution can pertain to numeric measures of interest (e.g., the completion time), to categorical outcomes (e.g., whether a given predicate will be fulfilled or violated), or to the sequence of future activities (and related payloads).

P

Overview

Predictive monitoring of business processes aims at providing predictions about the future of an ongoing (incomplete) process execution. The entry provides an overview of predictive process monitoring by introducing the dimensions that typically characterize existing approaches in the field, as well as using them to classify existing state-of-the-art approaches.

Introduction

Process mining deals with the analysis of business processes based on their behavior, observed

and recorded in *event logs*. In this context, an event log is a collection of execution *traces*, each representing one execution of the process (a.k.a. a *case*). Each trace consists of a sequence of time-stamped events, each capturing the execution of an activity. Each event may carry a data payload, i.e., a set of attribute-value pairs such as the resource(s) involved in the execution of the activity or other data recorded with the event.

Predictive business process monitoring (Maggi et al. 2014) is a branch of process mining that aims at predicting the future of an ongoing (uncompleted) process execution. Typical examples of predictions of the future of an execution trace relate to its completion time, to the fulfilment or violation of a certain predicate, or to the sequence of its future activities. Being able to predict in advance the time that a process instance will require to complete, or whether a predicate will be fulfilled, is of the utmost importance in different domains and scenarios, ranging from production to organizational processes. Indeed, this would allow organizations to prevent issues and delays and hence to save money, effort, and resources.

Predictive process monitoring approaches usually leverage past historical complete executions in order to provide predictions about the future of an ongoing case. They are typically characterized by two phases: a *training phase*, in which a predictive model is learned from historical traces, and a *prediction phase*, in which the predictive model is queried for predicting the future of an ongoing case.

Predictions in predictive business process monitoring approaches can pertain to different aspects or measures of interest related to the future of one or more process executions. Examples of outputs generated by predictive process monitoring approaches range from numeric or continuous measures as the remaining time or the overall cost of an ongoing execution, to categorical or boolean predictions, such as the risk associated to an ongoing execution or whether a given predicate will be fulfilled or violated, or even to the sequence of future activities characterizing a given case.

The rest of this entry is structured as follows: after a brief introduction about the importance of predictions in the context of business processes (section “[The Need of Predictions in Business Process Monitoring](#)”), the main features and dimensions characterizing predictive monitoring approaches for business processes are presented (section “[Predictive Process Monitoring Dimensions](#)”). The next three sections describe state-of-the-art approaches that produce, respectively, numeric predictions (section “[Numeric Predictions](#)”), predictions related to an outcome (section “[Outcome-Based Predictions](#)”), and predictions related to sequences of future activities (section “[Activity Sequence Predictions](#)”). Finally, section “[Multitype Predictions](#)” presents few frameworks collecting approaches providing more than one type of predictions, and section “[Conclusion](#)” concludes the entry.

The Need of Predictions in Business Process Monitoring

The capability to check the execution of business process instances in terms of correctness, as well as of timely completion, is an imperative for several organizations. Several research efforts have been carried out to address the problem of business process monitoring such as in Maggi et al. (2011b), Maggi et al. (2011a), Maggi et al. (2012), Ly et al. (2011), and Weidlich et al. (2011). Given a process model and a set of predicates, the aim is to monitor ongoing process executions in order to assess whether they comply with the predicates.

However, all these monitoring approaches are *reactive*. Indeed, the violation or the delay is identified only after its occurrence. Predicting the violation before would instead allow for supporting users in *preventing* it. We can think, for instance, about the preventive measures we would be able to take by knowing in advance that our train is delayed. Similarly, in the everyday life, organizations would get a number of benefits by knowing in advance about violations, deviances, and delays. This would indeed allow them to take preventive measures (e.g., reallocating resources)

in order to avoid the occurrence of those situations and, as often happens, to avoid money loss.

Predictive Process Monitoring Dimensions

Although predictive business process monitoring is a relatively young field, it has been growing fast in the latest years. The literature on predictive business process monitoring can be classified along three main dimensions:

- Type of prediction (i.e., the type of predictions provided as output).
- Type of adopted approach and technique.
- Type of information exploited in order to get predictions (i.e., the type of information taken as input).

Concerning the type of prediction, we can classify the existing prediction types into three main big categories:

- Predictions related to numeric or continuous measures of interest (*numeric predictions*). Typical examples in this setting are predictions related to the remaining time of an ongoing execution and predictions related to the duration or to the cost of an ongoing case.
- Predictions related to categorical or boolean outcomes (*outcome-based predictions*). Typical examples in this scenario are predictions related to the class of risk of a given execution or to the fulfilment of a predicate along the life cycle of a case.
- Predictions related to sequences of future activities (*activity sequence predictions*). Typical examples of predictions falling under this category refer to the prediction of the sequence of the future activities (and of their payload) of a process case upon its completion.

Predictive process monitoring approaches are usually characterized by two phases. In a first phase, the *training or learning* phase, one or

more prediction models are built or enriched by leveraging the information contained in the execution log. In the second phase, the *runtime or prediction phase*, the prediction model(s) is (are) exploited in order to get predictions related to one or more ongoing execution traces. We can identify two main groups of approaches dealing with the prediction problem:

- Approaches relying on an explicit model, e.g., annotated transition systems. The explicit model can either be discovered from the event log or enriched with the information the log contains, if it is already available.
- Approaches leveraging machine learning and statistical techniques, e.g., classification and regression models and neural networks. These approaches only rely on (implicit) predictive models built by encoding event log information in terms of features to be used as input for machine/deep learning techniques.

Finally, we can identify four different types of information that can be used as input to the predictive process monitoring approaches, e.g., for building a model annotated with execution information or for building the features to be used by machine learning approaches:

- Information related to the control flow, i.e., the sequence of events.
- Information related to the structured data payload associated to the events.
- Information related to unstructured (textual) content, which can be available together with the event log.
- Information related to process context, such as workload or resource availability.

In several approaches, more than one of these types of information is used in order to learn from the past.

In the following sections, the research works characterizing each of the three prediction type macro-categories, i.e., numeric, outcome-based, and activity sequence predictions, are presented. They are further classified based on the specific

output type and approach, as well as exploited input information.

Numeric Predictions

We can roughly classify the works dealing with numeric predictions in three groups, based on the specific type of predictions returned as output:

- Time predictions
- Cost predictions
- Generic performance predictions

Time predictions. The group of works focusing on the time perspective is a rich group. Several works, in this context, rely on explicit models. In van der Aalst et al. (2011), the authors present a set of approaches in which transition systems, which are built based on a given abstraction of the events in the event log, are annotated with time information extracted from the logs. Specifically, information about elapsed, sojourn, and remaining time is reported for each state of the transition system. The information is then used for making predictions on the completion time of an ongoing trace. Further extensions of the annotated transition system are proposed in Polato et al. (2014, 2018), where the authors apply machine learning techniques to the annotated transition system. In detail, in Polato et al. (2014), the transition systems are annotated with machine learning models such as naïve Bayes and support vector regression models. In Polato et al. (2018), instead, the authors present, besides two completely new approaches based on support vector regression, a refinement of the work in Polato et al. (2014) that takes into account also data. Moreover, the authors evaluate the three proposed approaches both on static and dynamic processes. Other extensions of the approach presented in van der Aalst et al. (2011) that also aim at predicting the remaining time of an ongoing trace, are the works presented in Folino et al. (2012, 2013). In these works the annotated transition system is combined with a context-driven predictive clustering approach. The idea behind predictive clustering

is that different scenarios can be characterized by different predictors. Moreover, contextual information is exploited in order to make predictions, together with control flow (Folino et al. 2012) or control flow and resources (Folino et al. 2013). Another approach based on the extraction of (explicit) models (*sequence trees*) is presented in Ceci et al. (2014) to predict the completion time (and the next activity) of the current ongoing case. Similarly to the predictive clustering approach, the sequence tree model allows for clustering traces with a similar sequence of activities (control flow) and to build a predictor model for each node of the sequence tree by leveraging data payload information. In Rogge-Solti and Weske (2013), the authors use generally distributed transition stochastic Petri nets (GDT-SPN) to predict the remaining execution time of a process instance. In detail, the approach takes as input a stochastic process model, which can be known in advance or inferred from historical data, an ongoing trace, and some other information as the current time in order to make predictions on the remaining time. In Rogge-Solti and Weske (2015), the authors also exploit the elapsed time since the last event in order to make more accurate predictions on the remaining time and estimating the probability to miss a deadline.

Differently from the previous approaches, in van Dongen et al. (2008), the authors only rely on the event log in order to make predictions. In detail, they develop an approach for predicting the remaining cycle time of a case by using nonparametric regression and leveraging activity duration and occurrence as well as other case-related data. In Bevacqua et al. (2013) and Cesario et al. (2016), the contextual clustering-based approach presented in Folino et al. (2012, 2013) is updated in order to address the limitation of the approaches based on transition systems, i.e., requiring that the analyst chooses the log abstraction functions. To this aim, the transition system predictor is replaced with standard regression algorithms. In Cesario et al. (2016), the clustering component of the approach is further improved in order to address scalability and accuracy issues. In Pandey et al. (2011) hidden Markov models

(HMM) are used for making predictions on the remaining time. A comparative evaluation shows that HMM provides more accurate results than annotated transition systems and regression models. In Senderovich et al. (2017) *inter-case feature predictions* are introduced for predicting the completion time of an ongoing trace. The proposed approach leverages not only the information related to the ongoing case but also the status of other (concurrent) cases (e.g., the number of concurrent cases) in order to make predictions. The proposed encodings demonstrated an improvement of the results when applied to two real-life case studies.

A prediction type which is very close to time but slightly different is the prediction of the delay of an ongoing case. In Senderovich et al. (2015), queuing theory is used to predict possible online delays in business process executions. The authors propose approaches that either enhance traditional approaches based on transition systems, as the one in van der Aalst et al. (2011), to take queueing effects into account or leverage properties of queue models.

Cost predictions. A second group of works focuses on cost predictions. Also in this group, we can find works explicitly relying on models as the work in Tu and Song (2016). In such a work, cost predictions are provided by leveraging a process model-enhanced cost (i.e., a frequent-sequence graph enhanced with cost) taking into account information about production, volume, and time.

Performance predictions. Finally, a third category of works deals with general numeric performance criteria, e.g., key performance indicators (KPIs). In Kang et al. (2011), the authors propose a predictive process monitoring technique, which starts by constructing a transition system from the event log. A state in this transition system represents the set of events that have occurred in the prefix of a case. Transitions are annotated with probabilities. The resulting transition system is used at runtime to predict key performance indicators of a running case.

The authors of the work in Zeng et al. (2008), use, instead, the ARIMA forecasting method for

aggregated key performance indicators rather than for metrics related to a single instance. In Castellanos et al. (2005), the authors present a business operation management platform equipped with forecasting functionalities. The platform allows for predictions of metric values related to a single running process instance (e.g., the completion time of the ongoing instance), as well as for predictions of aggregated values related to future instances (e.g., the number of orders that will be placed next Monday).

Outcome-Based Predictions

The second family of prediction approaches predicts outcomes related to an ongoing case. In this settings, two main specific types of predictions can be identified:

- Risk predictions
- Predicate fulfilment predictions

Risk predictions. A first large group of works falling under the umbrella of outcome-based predictions deals with the prediction of risks.

Also in this case, an important difference among state-of-the-art approaches is the existence of an explicit model guiding the prediction. For example, in Conforti et al. (2013), the authors present a technique for reducing process risks. The idea is supporting process participants in making risk-informed decisions, by providing them with predictions related to process executions. Decision trees are generated from logs of past process executions, by taking into account information related to data, resources, and execution frequencies provided as input with the process net. The decision trees are then traversed and predictions about risks returned to the users. In Conforti et al. (2015, 2016), two extensions of the work in Conforti et al. (2013) are presented. In detail, in the former, i.e., Conforti et al. (2015), the framework for risk-informed decisions is extended to scenarios in which multiple process instances run concurrently. Specifically, in order to deal with the risks related to different instances

of a process, a technique that uses integer linear programming is exploited to compute the optimal assignment of resources to tasks to be performed. In the latter, i.e., Conforti et al. (2016), the work in Conforti et al. (2013) is extended so that process executions are no more considered in isolation, but, rather, the information about risks is automatically propagated to similar running instances of the same process in real-time in order to provide early runtime predictions. In Metzger et al. (2015), three different approaches for the prediction of process instance constraint violation are investigated: machine learning, constraint satisfaction, and QoS aggregation. The authors, beyond demonstrating that all the three approaches achieve good results, identify some differences and propose their combination. Results on a real case study show that combining these techniques actually allows for improving the prediction accuracy.

Other works, devoted to risk prediction, do not take into account explicit models. For instance, in Pika et al. (2013a), the authors make predictions about time-related process risks by identifying and leveraging process risk indicators (e.g., abnormal activity execution time or multiple activity repetition) by applying statistical methods to event logs. The indicators are then combined by means of a prediction function, which allows for highlighting the possibility of transgressing deadlines. In Pika et al. (2013b), the authors extend their previous work by introducing a method for configuring the process risk indicators. The method learns from the outcomes of completed cases the most suitable thresholds for the process risk indicators, thus taking into account the characteristics of the specific process and hence improving the accuracy.

Predicate fulfilment predictions. A second group of predictions relates to the fulfilment of predicates. Almost all works falling under this category do not rely on any explicit model. For example, in Maggi et al. (2014) a framework for predicting the fulfilment (or the violation) of a predicate in an ongoing execution is introduced. Such a framework makes predictions by leveraging (i) the sequence of

events already performed in the case and (ii) the data payload of the last activity of the ongoing case. The framework is able to provide accurate results, although it demands for a high runtime overhead. In order to overcome such a limitation, the framework has been enhanced in Di Francescomarino et al. (2016b) by introducing a clustering preprocessing step in which cases sharing a similar behavior are clustered together. A predictive model – a classifier – for each cluster is then trained with the data payload of the traces in the cluster. In Di Francescomarino et al. (2016a), the framework is enhanced in order to support users by providing them with a tool for the selection of the techniques and the hyperparameters that best suit their datasets and needs. In Leontjeva et al. (2015), the authors consider traces as complex symbolic sequences, that is, sequences of activities each carrying a data payload consisting of attribute-value pairs. By starting from this assumption, the authors focus on the comparison of different feature encoding approaches, ranging from traditional ones, such as counting the occurrences of activities and data attributes in each trace, up to more complex ones, relying on hidden Markov models (HMM). In Verenich et al. (2016), the solution presented in Leontjeva et al. (2015) is enhanced with clustering, by proposing a two-phase approach. In the first phase, prefixes of historical cases are encoded as complex symbolic sequences and clustered. In the second phase a classifier is built for each of the clusters. At runtime, (i) the cluster closest to the current ongoing process execution is identified; and (ii) the corresponding classifier is used for predicting the process instance outcome (e.g., whether the case is normal or deviant). In Teinemaa et al. (2016), in order to improve prediction accuracy, unstructured (textual) information, contained in text messages exchanged during process executions, is also leveraged, together with control and data flow information. Specifically, different combinations of text mining (bag-of-n-grams, Latent Dirichlet Allocation and Paragraph Vector) and classification (random forest and logistic regression) techniques have been proposed and exercised. In Márquez-

Chamorro et al. (2017), an approach based on evolutionary algorithms is presented. The approach, which uses information related to a window of events in the event log, is based on the definition of process indicators (e.g., whether a process instance is completed in time, whether it is reopened). At training time, the process indicators are computed, the training event log encoded, and the evolutionary algorithms applied for the generation of a predictive model composed of a set of decision rules. At runtime, the current trace prefix is matched against the decision rules in order to predict the correct class for the ongoing running instance.

A special subclass of this category of works is the one related to papers focusing on the prediction of the violation of specific predicates. For instance, in Kang et al. (2012), the authors propose an approach for predicting abnormal termination of business processes at runtime and generating alarms for the early notification of probable abnormal terminations. They apply a fault detection technique based on the k-nearest neighbor algorithm to the attributes of process instances to estimate the probability that a fault occurs. In Cabanillas et al. (2014), an overall framework aiming at checking the safe execution of tasks and alerting in case of potential misbehaviors is presented. The prediction phase is only one of the framework phases, which also include the definition of the monitoring points and the expected behavior, the actual monitoring information collection, the event collection and normalization, and the deviance identification. Predictions are carried out by applying a support vector machine (SVM) approach to classify the successful completion of a process execution. To this aim both control and data flow information are taken into account.

Activity Sequence Predictions

A third more recent family of works deals with predicting the sequence of one, many, or all the future activities and their payload given the activities observed so far, as in Polato et al. (2018), Tax et al. (2017), Evermann et al. (2016),

and Di Francescomarino et al. (2017). In Polato et al. (2018), the authors propose an approach for predicting the sequence of future activities of a running case by relying on an annotated data-aware transition system, obtained as a refinement of the annotated transition system proposed in van der Aalst et al. (2011).

Other approaches, e.g., Evermann et al. (2016, 2017); Tax et al. (2017), make use of RNNs with LSTM cells. In particular, in Evermann et al. (2016, 2017) an RNN with two hidden layers trained with back propagation is presented, while in Tax et al. (2017), an LSTM and an encoding based on activities and timestamps is leveraged to provide predictions on the sequence of future activities and their timestamps. Finally, the work in Di Francescomarino et al. (2017) investigates how to take advantage of possibly existing a priori knowledge for making predictions on the sequence of future activities. To this aim, an LSTM approach is equipped with the capability of taking into account also some a priori sure knowledge.

Multitype Predictions

Some of the works available in the literature are general frameworks which are able to provide both numerical and categorical predictions. For instance the work in Leitner et al. (2013) describes a general framework for predicting the violation of service-level agreements (SLA) for business processes implemented as service composition by using data-driven statistical methods. Specifically, ANN-based regression is used for instance level violations, time series are used for aggregated metrics, and decision trees for nominal value predictions. The work presented in de Leoni et al. (2016) provides a general process mining framework which allows for correlating process and event characteristics (independent variables) with other characteristics (dependent variables). By collecting different types of approaches and techniques, it also allows for predicting different types of measures and outcomes. Finally, in Jorbina et al. (2017) the *Nirdizati* tool, a web-based tool for the generation of predictions

about running cases of a business process, is presented. Nirdizati, which is composed of a *training* and a *runtime* component, is able to continuously provide predictions of different types (both numerical and outcome-based) of performance indicators. In detail, the training component supports the users in the selection and tuning of the best predictive technique to deal with a specific dataset, by providing him/her with estimations of the accuracy of the results on a testing set. Once the most suitable technique has been selected, it is sent to the runtime component which uses it for continuously monitoring the ongoing executions and providing predictions related to the desired performance indicator at runtime.

Conclusion

Predictive monitoring of business processes has been gaining importance and raising more and more interest within the scientific community. This entry provides an overview of the literature related to the predictive business process monitoring. The main concepts related to the topic, as well as the main dimensions (e.g., the type of prediction, approach, and information exploited for providing predictions) characterizing the works related to the predictive monitoring of business processes, have been identified and presented. The main research works populating the literature related to such an emerging field have been classified and described.

Cross-References

- [Business Process Deviance Mining](#)
- [Queue Mining](#)

References

Bevacqua A, Carnuccio M, Folino F, Guarascio M, Pontieri L (2013) A data-adaptive trace abstraction approach to the prediction of business process performances. In: Hammoudi S, Maciaszek LA, Cordeiro J, Dietz JLG (eds) ICEIS (1), SciTePress, pp 56–65

- Cabanillas C, Di Cicco C, Mendling J, Baumgrass A (2014) Predictive task monitoring for business processes. Springer International Publishing, Cham, pp 424–432. https://doi.org/10.1007/978-3-319-10172-9_31
- Castellanos M, Salazar N, Casati F, Dayal U, Shan MC (2005) Predictive business operations management. Springer, Berlin/Heidelberg, pp 1–14. https://doi.org/10.1007/978-3-540-31970-2_1
- Ceci M, Lanotte PF, Fumarola F, Cavallo DP, Malerba D (2014) Completion time and next activity prediction of processes using sequential pattern mining. Springer International Publishing, Cham, pp 49–61. https://doi.org/10.1007/978-3-319-11812-3_5
- Cesario E, Folino F, Guarascio M, Pontieri L (2016) A cloud-based prediction framework for analyzing business process performances. Springer International Publishing, Cham, pp 63–80. https://doi.org/10.1007/978-3-319-45507-5_5
- Conforti R, de Leoni M, La Rosa M, van der Aalst WMP (2013) Supporting risk-informed decisions during business process execution. In: Proceeding of CAiSE 2013. Springer, pp 116–132
- Conforti R, de Leoni M, La Rosa M, van der Aalst WMP, ter Hofstede AHM (2015) A recommendation system for predicting risks across multiple business process instances. Decis Support Syst 69:1–19. <https://doi.org/10.1016/j.dss.2014.10.006>
- Conforti R, Fink S, Manderscheid J, Röglinger M (2016) PRISM – a predictive risk monitoring approach for business processes. Springer International Publishing, Cham, pp 383–400. https://doi.org/10.1007/978-3-319-45348-4_22
- de Leoni M, van der Aalst WMP, Dees M (2016) A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf Syst 56:235–257. <https://doi.org/10.1016/j.is.2015.07.003>
- Di Francescomarino C, Dumas M, Federici M, Ghidini C, Maggi FM, Rizzi W (2016a) Predictive business process monitoring framework with hyperparameter optimization. In: Advanced information systems engineering – Proceedings of 28th international conference, CAiSE 2016, Ljubljana, 13–17 June 2016, pp 361–376. https://doi.org/10.1007/978-3-319-39696-5_22
- Di Francescomarino C, Dumas M, Maggi FM, Teinemaa I (2016b) Clustering-based predictive process monitoring. IEEE Trans Serv Comput PP(99)
- Di Francescomarino C, Ghidini C, Maggi FM, Petrucci G, Yeshchenko A (2017) An eye into the future: leveraging A-priori knowledge in predictive business process monitoring. Springer International Publishing, Cham, pp 252–268. https://doi.org/10.1007/978-3-319-65000-5_15
- Evermann J, Rehse JR, Fettke P (2016) A deep learning approach for predicting process behaviour at runtime. In: PRAISE-2016
- Evermann J, Rehse JR, Fettke P (2017) Predicting process behaviour using deep learning. Decis Support Syst. <https://doi.org/10.1016/j.dss.2017.04.003>

- Folino F, Guarascio M, Pontieri L (2012) Discovering context-aware models for predicting business process performances. In: Proceeding of on the move to meaningful internet systems (OTM). Springer, pp 287–304
- Folino F, Guarascio M, Pontieri L (2013) Discovering high-level performance models for ticket resolution processes. Springer, Berlin/Heidelberg, pp 275–282. https://doi.org/10.1007/978-3-642-41030-7_18
- Jorbina K, Rozumnyi A, Verenich I, Di Francescomarino C, Dumas M, Ghidini C, Maggi FM, La Rosa M, Raboczi S (2017) Nirdizati: a web-based tool for predictive process monitoring. In: Proceedings of the BPM demo track and BPM dissertation award co-located with 15th international conference on business process modeling (BPM 2017), Barcelona, 13 Sept 2017
- Kang B, Jung J, Cho NW, Kang S (2011) Real-time business process monitoring using formal concept analysis. Ind Manag Data Syst 111(5):652–674. <https://doi.org/10.1108/02635571111137241>
- Kang B, Kim D, Kang SH (2012) Real-time business process monitoring method for prediction of abnormal termination using knni-based lof prediction. Expert Syst Appl 39(5):6061–6068. <https://doi.org/10.1016/j.eswa.2011.12.007>
- Leitner P, Ferner J, Hummer W, Dustdar S (2013) Data-driven and automated prediction of service level agreement violations in service compositions. Distrib Parallel Databases 31(3):447–470. <https://doi.org/10.1007/s10619-013-7125-7>
- Leontjeva A, Conforti R, Di Francescomarino C, Dumas M, Maggi FM (2015) Complex symbolic sequence encodings for predictive monitoring of business processes. In: BPM 2015. Springer International Publishing, pp 297–313
- Ly LT, Rinderle-Ma S, Knuplesch D, Dadam P (2011) Monitoring business process compliance using compliance rule graphs. In: CoopIS, pp 82–99
- Maggi FM, Montali M, Westergaard M, van der Aalst WMP (2011a) Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Proceeding of BPM 2011
- Maggi FM, Westergaard M, Montali M, van der Aalst WMP (2011b) Runtime verification of LTL-based declarative process models. In: Proceeding of RV, vol 7186, pp 131–146
- Maggi FM, Montali M, van der Aalst WMP (2012) An operational decision support framework for monitoring business constraints. In: FASE12
- Maggi FM, Di Francescomarino C, Dumas M, Ghidini C (2014) Predictive monitoring of business processes. In: Advanced information systems engineering – Proceedings of 26th international conference, CAiSE 2014, Thessaloniki, 16–20 June 2014, pp 457–472
- Márquez-Chamorro AE, Resinas M, Ruiz-Cortés A, Toro M (2017) Run-time prediction of business process indicators using evolutionary decision rules. Expert Syst Appl 87(Suppl C):1 – 14. <https://doi.org/10.1016/j.eswa.2017.05.069>
- Metzger A, Leitner P, Ivanović D, Schmieders E, Franklin R, Carro M, Dustdar S, Pohl K (2015) Comparing and combining predictive business process monitoring techniques. IEEE Trans Syst Man Cybern Syst 45(2):276–290. <https://doi.org/10.1109/TSMC.2014.2347265>
- Pandey S, Nepal S, Chen S (2011) A test-bed for the evaluation of business process prediction techniques. In: 7th international conference on collaborative computing: networking, applications and worksharing (CollaborateCom), pp 382–391. <https://doi.org/10.4108/icst.collaboratecom.2011.247129>
- Pika A, van der Aalst WMP, Fidge CJ, ter Hofstede AHM, Wynn MT (2013a) Predicting deadline transgressions using event logs. Springer, Berlin/Heidelberg, pp 211–216. https://doi.org/10.1007/978-3-642-36285-9_22
- Pika A, van der Aalst WMP, Fidge CJ, ter Hofstede AHM, Wynn MT (2013b) Profiling event logs to configure risk indicators for process delays. Springer, Berlin/Heidelberg, pp 465–481. https://doi.org/10.1007/978-3-642-38709-8_30
- Polato M, Sperduti A, Burattin A, de Leoni M (2014) Data-aware remaining time prediction of business process instances. In: 2014 international joint conference on neural networks (IJCNN), pp 816–823. <https://doi.org/10.1109/IJCNN.2014.6889360>
- Polato M, Sperduti A, Burattin A, de Leoni M (2018) Time and activity sequence prediction of business process instances. Computing. <https://doi.org/10.1007/s00607-018-0593-x>
- Rogge-Solti A, Weske M (2013) Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: ICSOC 2013. Springer, pp 389–403
- Rogge-Solti A, Weske M (2015) Prediction of business process durations using non-markovian stochastic petri nets. Inf Syst 54(Suppl C):1–14. <https://doi.org/10.1016/j.is.2015.04.004>
- Senderovich A, Weidlich M, Gal A, Mandelbaum A (2015) Queue mining for delay prediction in multi-class service processes. Inf Syst 53:278–295. <https://doi.org/10.1016/j.is.2015.03.010>
- Senderovich A, Di Francescomarino C, Ghidini C, Jorbina K, Maggi FM (2017) Intra and inter-case features in predictive process monitoring: a tale of two dimensions. Springer International Publishing, Cham, pp 306–323. https://doi.org/10.1007/978-3-319-65000-5_18
- Tax N, Verenich I, La Rosa M, Dumas M (2017) Predictive business process monitoring with LSTM neural networks. In: Advanced information systems engineering – Proceedings of 29th international conference, CAiSE 2017, Essen, 12–16 June 2017, pp 477–492
- Teinemaa I, Dumas M, Maggi FM, Di Francescomarino C (2016) Predictive business process monitoring with structured and unstructured data. In: BPM 2016, pp 401–417
- Tu TBH, Song M (2016) Analysis and prediction cost of manufacturing process based on process

- mining. In: 2016 international conference on industrial engineering, management science and application (ICIMSA), pp 1–5. <https://doi.org/10.1109/ICIMSA.2016.7503993>
- van der Aalst WMP, Schonenberg MH, Song M (2011) Time prediction based on process mining. Inf Syst 36(2):450–475
- van Dongen BF, Crooy RA, van der Aalst WMP (2008) Cycle time prediction: when will this case finally be finished? Springer, Berlin/Heidelberg, pp 319–336. https://doi.org/10.1007/978-3-540-88871-0_22
- Verenich I, Dumas M, La Rosa M, Maggi FM, Di Francescomarino C (2016) Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. Springer International Publishing, Cham, pp 218–229. https://doi.org/10.1007/978-3-319-42887-1_18
- Weidlich M, Ziekow H, Mendling J, Günter O, Weske M, Desai N (2011) Event-based monitoring of process execution violations. In: Proceeding of CAiSE
- Zeng L, Lingenfelder C, Lei H, Chang H (2008) Event-driven quality of service prediction. Springer, Berlin/Heidelberg, pp 147–161. https://doi.org/10.1007/978-3-540-89652-4_14

Predictive Monitoring of Business Processes

► Predictive Business Process Monitoring

Privacy Cube

Jhalak Hota¹, Deepak Puthal², and Abhay Kumar Samal³

¹Trident Academy of Technology, Bhubaneswar, India

²Faculty of Engineering and Information Technologies, School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW, Australia

³Trident Academy of Technology, Bhubaneswar, India

Synonyms

Sentiment-based privacy preserving data publishing techniques for social networks

Definitions

Actor: The *individual* or *organizations* involved in a social network.

Post: Thoughts expressed by the actors.

People: Either a group of people who are the author or about the people mentioned in the post.

Place: Place of origin or about what the post is. Place of interest of the individual or where they belongs to.

Time: Time of the post or the time frame the post talks about. Time frame of the people involved.

Privacy coordinate: A Cartesian coordinate system that uses people, place, and time as its axes. These three parameters act like the *basis* through which post/actors are defined.

Sensitivity: Vulnerability due to social, religious, or some other factors.

Sentiment: A view or opinion that is held or expressed.

Privacy in Social Networks

Web 2.0 provides a two-way knowledge and information sharing platform. Many of these information of the Web 2.0 may lead different types of interpretation and perception by the audience or the consumer of this information. The biggest challenge is that the content creator has no control over this *response*. This response may be positive or negative and sometimes very frustrating. Apart from this the social media is prone to different kinds of privacy threats mentioned in these survey papers (Díaz and Ralescu 2012; Gunatilaka 2011). Among privacy issues like de-anonymization attack and neighborhood attack, the biggest threat is the threat of privacy leakage, which is caused by leakage through poor privacy setting or leakage of information to third-party application or leakage of information to third-party domain. Identity theft, spam, and malware issues are quite prevalent in social media.

Problem Formulation

Sensitivity and Privacy

Sensitivity is one of the most important social virtues, and this sensitivity has no formal representation yet. Some information may be very sensitive to a group of people and for some it is absolute okay. Closely monitoring, a general observation came that this variation is due to three main parameters:

- **People:** Who they are, their belief, and their ideology and in case of an event who are and what kind of people are involved.
- **Time:** What is the time frame of occurrence of event or time period in which the people lived.
- **Place:** Where the event has taken place or where the people belongs to.

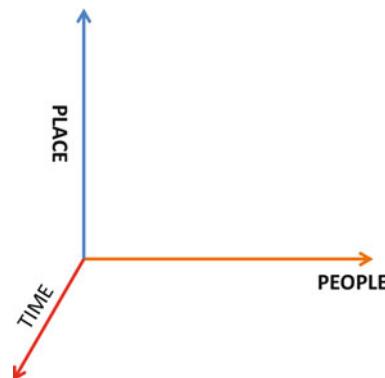
Modelling Sensitivity

Sensitivity can be modelled as score ranging from -1 to $+1$ through 0 . This score is an indicator of how individuals or an event and individual align themselves in a coordinate system whose axes are **people, place, and time** as shown in Fig. 1. This three-dimensional vector simulates both the social actors and information.

Algorithmic Framework

Core Idea

Privacy cube approach is based on the core Ideas that “*Only Relevant Information should be Displayed irrespective of Friends sharing it or not & Information and People should be treated alike*” which is an extension to the approach mentioned in Babu et al. (2013, 2014). In order to achieve this goal, we need to find a common *basis* that a social network post and a social actor share. Three of such basis are (1) people, (2) place, and (3) time. A person is known from the people he knows, places he has interest or he belongs in, and time period he belongs to. Similar thing can be said about the information too. Information are about these three or involve these three. Thus we are able to map the two differently perceived



Privacy Cube, Fig. 1 Privacy coordinate

objects into a common *Basis*. This makes evaluation of similarity between a social actor and information more effective.

Scoring Mechanism: SentSim

SentSim is a scoring mechanism that indicates the *sentimental similarity* between two actors, or an actor and information is proposed. This metric measures the *cosine of the angle between* two *social vectors* (social actors and social post). Table 1 explains our scoring mechanism as follows:

P

Flowchart

A detail flowchart explains the process given in Fig. 2. The following points explain the terms used in the flow chart.

POST The social network post or actor that we are going to, which is the input to the system.

TAGS Tags are the provision through which the author provides information about the people, place, and time involved.

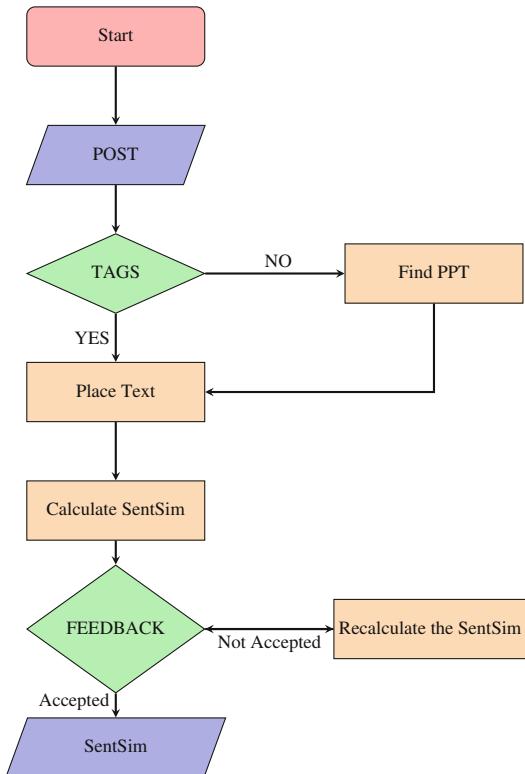
FindPPT This module is responsible for extracting name of the people, place, and time using natural language processing techniques from the post.

PlaceText Use the extracted people, place, and time to place the information in the privacy coordinate.

FEEDBACK It is a mechanism through which the calculated *SentSim* is verified of the information under consideration. Thresholds

Privacy Cube, Table 1 Scoring mechanism

SentSim	Geometry	Inference
+1	Both the vectors are aligned in the same direction as angle between them is 0	The information may have highest positive influence on the actor
0	Both the vectors are orthogonal to each other as angle between them is 90	The information has no or minimum influence on the actor
-1	Both vectors are aligned in opposite direction as angle between them is 180	The information may have highest negative influence on the actor

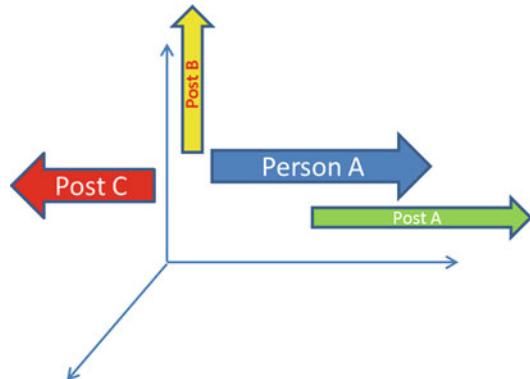
**Privacy Cube, Fig. 2** Flow chart describing the process

were to be fixed. If the $|SentSim_i - SentSim_{i+1}| \leq Threshold$, the score is accepted. If not then the iteration runs again.

Example

Consider the scenario explained in Fig. 3.

Scenario 1 *Post A* is sentimentally aligned with *Person A*, as the angle between them is zero. In this case Person A is very much interested

**Privacy Cube, Fig. 3** Example explaining the work of privacy cube

in Post A. *Post A* must be placed high up in the time line of Person A

Scenario 2 *Post B* is vertically aligned with *Person A*. The angle between them is 90. *Post B* is neutral to Person A. So there is a least chance that post be sentimentally hurt Person A.

Scenario 3 *Post C* is sentimentally aligned in opposite direction. This post has the highest probability to hurt the sentiment or breach the privacy of the persons involved in Post C. This post must not be displayed to Person A

Conclusion and Future Work

Though privacy and sentiment seem not to be correlated, in fact they are very much related virtues of human spirit. People love to be with the people with whom they share common sentimental values, which this model ensures through geometrical interpretation. This plays an important role in preserving privacy of individuals. This works so far based on this philosophy. By categorizing

social actor and social post, we can better the experience of social networking.

The advantage of privacy cube is that it brought both social actor and social post into common basis, i.e., people, place, and time. Due to this any analysis to be done would have more accuracy if we use some different basis for actors and posts. However like all method, *privacy cube* has some limitations which primarily due to its dependence on other modules, like *FindPPT* and calculation of *SentSim*. Future work is concentrated on the following area:

- *Proper noun classification* An efficient mechanism needs to be formulated that is capable of handling ever-changing social lingo.
- Developing a mechanism to calculate the sentimental similarity of *SentSim*. This can be done by developing a mechanism that studies the sentimental pattern of a group of people based upon their privacy axes. Accordingly it tries to predict the sentimental similarity of the actor or post in question.
- Developing a *Feedback* mechanism that perfectly suit these kind of environment.

References

- Babu KS, Jena SK, Hota J, Moharana B (2013) Anonymizing social networks: a generalization approach. *Comput Electr Eng* 39(7):1947–1961
- Babu KS, Hota J, Jena SK (2014) Privacy preserving social networking. *Int J Comput Sci Eng* 9(3): 165–176
- Díaz I, Ralescu A (2012) Privacy issues in social networks: a brief survey. *Adv Comput Intell* 300:509–518
- Gunatilaka D (2011) A survey of privacy and security issues in social networks. In: Proceedings of the 27th IEEE international conference on computer communications. IEEE Computer Society, Washington

Privacy Protection via Replacing Identifiers with Domain Pseudonyms

- Privacy-Aware Identity Management

Privacy, Authentication, Integrity, Nonrepudiation (PAIN)

- Security and Privacy in Big Data Environment

Privacy-Aware Identity Management

Mirosław Kutyłowski and Przemysław

Błaśkiewicz

Faculty of Fundamental Problems of Technology, Wrocław University of Science and Technology; National Science Center, Wrocław, Poland

Synonyms

Authenticating anonymized data with domain signatures; Privacy protection via replacing identifiers with domain pseudonyms

Definitions

P

Definition 1 *Personal data protection* means measures aiming to guarantee that processing of data concerning identified or identifiable persons (data subjects):

- takes place based on a consent of the data subject or is otherwise explicitly permitted by law,
- is confined solely to the purpose of processing,
- is secure by design and by default:

whereas *processing* is any operation on personal data, including in particular storing, removing, and giving access to it.

Definition 2 *Pseudonimization* is a process of translating identifiers into pseudonyms and replacing the identifiers in a data set with the corresponding pseudonyms. A pseudonimization

process should be irreversible, except for a deanonymization procedure (if defined).

Deanonymization is a process which links back the pseudonyms with the identifiers. It should require the use of appropriate cryptographic secret keys.

Definition 3 A *domain* is a virtual area of activity, where all data concerning one physical person should be linked and be available as corresponding to the same person.

Definition 4 A *domain pseudonyms scheme* is a pseudonymization scheme such that an identifier pointing to a particular person is mapped to a set of *different* pseudonyms, one per each domain where the person may be present. The pseudonyms in different domains should be *unlinkable*.

Definition 5 A domain pseudonym scheme fulfills the *unlinkability* property, if it is impossible to distinguish with a non-negligible advantage between the two sets of pseudonyms:

- the pseudonyms generated according to the scheme for any one domain,
- the pseudonyms generated independently at random for each identifier and all other domains.

Definition 6 By a *domain signature scheme*, we mean a digital signature scheme, where:

- the signature points to a domain pseudonym and not to the identity of the signatory,
- a signatory uses a single private key for creating signatures in all domains,
- verification process is based on the domain pseudonym, signature, signed data, domain parameters, and, optionally, additional domain-specific data of the signatory.

If the scheme supports revocation, then the verification process uses current revocation data.

Definition 7 *Unlinkability* of a domain signature scheme is satisfied, if conditions of Definition 5 are fulfilled, but where the distinguisher may request also signatures corresponding to pseudonyms and where in case of the second situation the signing key is chosen independently at random for each domain.

Definition 8 By *revocation*, we mean a process that enables declaration of certain domain pseudonyms as *revoked*. Thereafter, the verification of a signature corresponding to any of these pseudonyms should yield the result invalid.

Overview

This chapter provides a concise discussion on ways of obtaining privacy of users' digital activities by means of pseudonymization schemes and their application in activity domains.

Legal rationale behind the introduction of these mechanisms is provided along with definitions of elementary concepts of such systems as domain pseudonyms and pseudonymous signatures. A number of possible solutions are provided along with references to already published papers on the subject.

GDPR Regulation of European Union

GDPR Scope

The General Data Protection Regulation (GDPR) of The European Parliament and the Council of the European Union (2016) establishes strict rules concerning processing of personal data, where:

personal data – is understood as any data concerning identified or identifiable person.

processing – means any operation on personal data, in particular:

collection, recording, organisation, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available,

alignment or combination, restriction, erasure or destruction.

The GDPR regulation applies to the processing of personal data by automated means (hence in practically all IT systems) and in filing systems. There are a few exceptions (e.g., it does not cover processing for purely personal use and by law enforcement bodies); however, profiling users does fall into the scope of the GDPR.

The GDPR regulation applies to:

- any processing that is related to the activities (of the processor) performed in the European Union,
- processing data of persons in the European Union, if it is related to the offering of goods or services as well as profiling such users.

The regulation applies regardless of the location where the processing is executed – e.g., profiling European customers by an establishment in the USA falls into the scope of the regulation.

Thereby, in most cases big data processing is likely to fall into the scope of the GDPR regulation, and the data processor is obliged by law to conform with its rules.

GDPR Rules

There are the following fundamental principles of personal data processing:

lawfulness, fairness, and transparency – processing requires a legal basis (in most cases an explicit consent of the data subject); it should be fair and transparent to the data subject; in particular, the data subject may request access to information concerning processing their data;

purpose limitation – personal data may be collected and processed only for specified, explicit, and legitimate purposes; in particular, changing the purpose and scope of processing is forbidden (without the subject's explicit consent);

data minimization – processing must be adequate, relevant, and limited to the purpose of processing; in particular, if there is a legal

basis for processing, its goal must be achieved with minimal amount of personal data;

accuracy – processing must be accurate; the data must be kept up to date (if necessary, in other cases, see point below); in particular, the processing party must provide means for the data subject to correct their data;

storage limitation – personal data should be kept:

in a form which permits identification of data subjects for no longer than is necessary for the purposes for which the personal data are processed

integrity and confidentiality – the party processing personal data has to ensure:

appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures

accountability – compliance with the rules of GDPR should be demonstrated. The system used for processing must be secure by design.

One of the detailed rights is the right-to-be-forgotten: at any time, a data subject may withdraw his consent to process their personal data.

P

Pseudonymization

Following the rules of the GDPR regulation may require a lot of effort. One of the pragmatic approaches is to:

- change the status of personal data to non-personal data.

Such a conversion occurs, whenever

- it is no longer possible to link the data to a physical person.

As a minimum, such a process requires a pseudonymization – the conversion of all identifiers of a physical person into pseudonyms.

This concerns both the explicit identifiers (such as name, personal ID number, etc.) and the implicit identifiers (such as a public key, postal address, non-anonymous email address, unique position, artistic pseudonym, etc.). Of course, pseudonymization is only a necessary condition and not a sufficient one. Nevertheless, pseudonymization is explicitly pointed to by the GDPR regulation. It states among others that:

the [data] controller should adopt the principles of data protection by design and data protection by default. Such measures could consist, inter alia, of minimising the processing of personal data, pseudonymising personal data as soon as possible

In order to be effective, derivation of pseudonyms must be automated. The following basic methods can be applied:

1. encryption of each identifier I by its ciphertext $Enc_K(I)$, where K is the private key of the party performing pseudonymization,
2. converting each identifier I by $F(I, K)$, where F is a one-way function and K is a secret key.
3. converting each identifier I by $F(I)$, where F is a one-way function.

The first method is reversible: if necessary, the party holding the key K can convert the pseudonym back to its corresponding identifier. This can both be an advantage and disadvantage: for the party holding the key K , the data have still the status of personal data, as the physical person is still identifiable by this party. This enforces on the party application of adequate measures for processing the data. On the other hand, if data may be used as personal data in the future, then reversibility is a necessary condition.

The second method can be used whenever data should permanently lose the status of personal data and the data set should undergo as minimal changes as possible.

The third method has the additional advantage that many parties may perform anonymization of different parts of the same data set, while the records concerning the same person will remain linkable. On the downside, after pseudonymization, everybody can check whether a pseudonym

P corresponds to an identifier I by checking whether $P = F(I)$. Therefore this option has very limited applicability.

Domains

By definition, pseudonymization is a process of replacing an identifier with a pseudonym. However, it does not mean that one identifier should correspond always to the same pseudonym. If that were the case, then such unique mapping might facilitate the linking of a pseudonym to its original identifier without breaking the cryptographic mechanism used for generating pseudonyms. The attacker, in the process called profiling, might analyze all data corresponding to a single pseudonym and create a set of features that uniquely define one particular person, thus revealing the identity of the pseudonym owner.

Following the data minimization principle of GDPR and aiming to reduce the chances of the above mentioned attack,

- an identifier I should be replaced by the same pseudonym only when it is necessary to maintain the information that such anonymized data correspond to the same (anonymous) person.

The concept of domains is a framework that aims to achieve this goal.

Separation Concept

The following assumptions about the information processing model are made:

- there are separate (virtual) domains of data processing,
- data concerning one person should be linkable within a domain,
- linking data from different domains but concerning the same person should be prevented.

The model originates from Austrian Bürgerkarte and separation of domains in the public administration of that country.

Domain-Specific Pseudonyms

The following rules are set:

1. for each identity there is a single pseudonym per domain (domain-specific pseudonym),
2. each user has a single identity and a single private key,
3. in order to generate a domain-specific pseudonym, it is necessary to use the private key.

Domain-Specific Signatures

A domain signature scheme is a digital signature scheme, where:

- each signature corresponds to a single domain and a single domain pseudonym,
- each user holds a single private key and uses it for creating his domain pseudonyms and domain signatures in all domains,
- the verification process refers to a domain pseudonym instead of the real identity of the signatory, as well as domain parameters.

Deanonymization

For various reasons it might be necessary to trace back the links between the real identity and pseudonym(s) or between pseudonyms of the same person. This may happen, for instance, due to law enforcement (linking a pseudonym with a real identity) or merging two domains after merging two web services (linking pseudonyms in two domains in order to merge accounts).

There are the following basic use cases for deanonymization procedures:

- find the domain pseudonym for a given real identity and a domain,
- find the real identity corresponding to a given domain pseudonym,
- given a domain pseudonym from domain A , find the domain pseudonym of the same person for domain B .

By definition, such operations can be easily done by the owner of the pseudonym(s) by applying adequate pseudonym generation

algorithms. However, for practical reasons, the deanonymization scheme must also include a scenario where the pseudonym owner is not involved in the procedure. And in order to prevent deanonymization by non-authorized parties, the procedure should not be possible without certain secret keys remaining solely in the possession of authorized entities.

Security Requirements

Seclusiveness

One of the basic properties of domain pseudonyms is the rule stating that *each user has one pseudonym per domain*. In order to fulfill this property in organizational way, the following architecture is adopted:

- a person can be enabled to join the system only with the consent of the Issuer, after authenticating his identity (joining procedure),
- the Issuer enables only one private key per physical person.

The joining procedure comprises in particular of:

- generation of a private key of the person (either by the Issuer, or the joining person, or by both of them),
- creating cryptographic material (if necessary), which can be used to show that the private key has been confirmed by the Issuer.

The *seclusiveness* property means that the following statements always hold for any valid pseudonym I for domain D :

- there exists a real person A that went through the joining procedure, and ...
- the pseudonym I is equal to the pseudonym of A in domain D created according to the joining procedure.

In order to break the seclusiveness property, the adversary may collude with the persons that

have already joined the system and in particular have access to their cryptographic material.

Solution Examples

Domain Identifiers

For all solutions discussed below, there is a unique public key associated with each domain. Let PK_D^{sector} denote the public key for sector D . Some authors call it *domain parameter*, as there might be no private key corresponding to PK_D^{sector} .

Below we present two options which can be traced back to the BSI (2016) Technical Recommendation (and its former versions).

Ad Hoc Domains

An ad hoc domain does not require management activities in order to create a domain. The public key is typically generated as

$$PK_D^{\text{sector}} := \text{Hash}(D),$$

where D is the official domain identifier and Hash is a hash function mapping into a group used for generating pseudonyms.

Additional requirement is that the discrete logarithm of PK_D^{sector} is not publicly known. It is easy to satisfy this constrain. For example, deriving Hash function from the SHA family assures that this requirement holds (i.e., it is infeasible to derive the discrete logarithm of PK_D^{sector}). Similarly, one can use the Pedersen hash Pedersen (1991) in the following way:

$$PK^{\text{sector}} := g^{\text{Hash}_1(D)} \cdot \beta^{\text{Hash}_2(D)}$$

where the discrete logarithm of β with respect to g is unknown (except for an authority or authorities performing deanonymization) and Hash_1 and Hash_2 are independent hash functions. The above computations are performed in a group, where at least the Discrete Logarithm Problem is hard.

Created Domains

In this case a domain emerges through some activity of an authority creating domains. As a

result, the existence of a domain is somehow confirmed, e.g., via:

trusted list – a (white)list of all domains registered. The list should be authenticated via a digital signature, analogously to CRL lists.
certificate – a certificate confirming PK_D^{sector} .

The second option has been used in Germany for domains defined for personal identity cards (*Berechtigungszertifikat*) and Restricted Identification scheme BSI (2016).

Domain Pseudonyms

Pseudonyms Based on DH Problem

The most popular method of generating domain pseudonyms for a user holding a secret key x is to derive

$$I := (PK_D^{\text{sector}})^x.$$

Given a public key $y = g^x$ of a person holding x and a domain pseudonym I , it should be infeasible to determine whether $I = (PK_D^{\text{sector}})^x$, unless the discrete logarithm of PK_D^{sector} is known to the verifier. Answering this question is related to solving the decisional Diffie-Hellman Problem (and, more directly, solving the DH linking problem Kutyłowski et al. 2011).

German Restricted Identification

The Restricted Identification scheme implemented on German ID cards uses a slightly different version of pseudonyms (BSI 2016). Namely, the pseudonym of a user holding key x is computed as

$$\text{Hash}\left((PK_D^{\text{sector}})^x\right).$$

Introducing the hash function is an additional security measure, since a party receiving the pseudonym does not learn the input for the DDH problem. So if two sectors have public keys $PK_{D_1}^{\text{sector}}$ and $PK_{D_2}^{\text{sector}}$ and somehow learn r such that $(PK_{D_1}^{\text{sector}})^r = PK_{D_2}^{\text{sector}}$, then they still cannot convert the pseudonyms from D_1 to pseudonyms in D_2 .

Computing the Pseudonyms by the Domain Manager

When a domain manager knows the discrete logarithm r of the domain's public key, $PK_D^{sector} = g^r$, then it can create domain pseudonyms based on user's public key $y = g^x$:

$$I := y^r \quad (\text{1st scheme})$$

or

$$I := \text{Hash}(y^r) \quad (\text{2nd scheme}).$$

Thereby, the user does not need to provide his pseudonyms: it can be computed later by the domain manager and the pseudonymization may be performed without interacting with the user.

Authenticating Data with Domain Signature

Whitelist-Based Scheme

The simplest scheme may be based on replacing the base element with PK_D^{sector} in DSA signature. The user holds a sector key x and his domain pseudonym is $(PK_D^{sector})^x$. The domain signature for data m is defined as usual for DSA:

- choose $k \in \mathbb{Z}_q$ at random,
- compute

$$r := ((PK_D^{sector})^k \bmod p) \bmod q,$$
- compute

$$s := k^{-1}(\text{Hash}(m) + xr) \bmod q,$$

The verification procedure is the same as for DSA, except that the generator g is now replaced by PK_D^{sector} .

The main problem with this solution is that seclusiveness is achieved by whitelist generation separately for each domain: given user's public key $y = g^x$, the entry $(PK_D^{sector})^x$ is generated as

$$y^d,$$

where $PK_D^{sector} = g^d$. Value of d can be distributed among two or more authorities. If $PK_D^{sector} = g^{d_1 \cdot d_2}$, then the authority holding d_1

computes $y_1 := y^{d_1}$ and the second authority computes $y_2 := y_1^{d_2}$.

Application scope of this scheme is limited to specific applications where the set of domain identifiers is closed.

German Pseudonymous Signature

The pseudonymous signature scheme Bender et al. (2012) is closely related to Okamoto signatures. It is based on Discrete Logarithm Problem, and in particular it requires neither pairings nor any kind of certificate, whitelist, etc. The price paid is the requirement for unconditionally secure hardware devices implementing signature keys.

There are two system secrets in this scheme, SK_{ICC} and $SK_M < q$, where q is the (prime) order of the group used. The corresponding public keys are defined as follows:

$$PK_{ICC} = g^{SK_{ICC}}, \quad PK_M = g^{SK_M}.$$

A user joining the system gets secrets x_0, x_1 , where x_1 is chosen at random and x_0 satisfies the quality:

$$SK_{ICC} = x_0 + x_1 \cdot SK_M \quad \bmod q.$$

The corresponding public keys are $P_0 = g^{x_0}$ and $P_1 = g^{x_1}$, and they are used exclusively in the case of deanonymization.

The domain public key PK_D^{sector} is created as g^d (or $g^{d_1 \cdot d_2}$ if the secret d has to be split).

A user holding keys x_0, x_1 has *two* valid domain pseudonyms (in each domain). They are defined as

$$\begin{aligned} I_{D,0} &= (PK_D^{sector})^{x_0} \\ I_{D,1} &= (PK_D^{sector})^{x_1}. \end{aligned}$$

A signature over m for a domain with the public key PK_D^{sector} is created as follows:

1. choose at random $k_0, k_1 < q$,
2. compute $Q := g^{k_0} \cdot PK_M^{k_1}$,
3. compute:

$$A_0 := (PK_D^{sector})^{k_0},$$

$$A_1 := (PK_D^{sector})^{k_1},$$

4. compute $c := \text{Hash}(Q, I_{0,D}, A_0, I_{1,D}, A_1, PK_D^{\text{sector}}, m)$,
5. compute:
 $s_0 := k_0 - c \cdot x_0 \bmod q,$
 $s_1 := k_1 - c \cdot x_1 \bmod q.$

The resulting signature is

$$\sigma(m) = (c, s_0, s_1).$$

The verification process for a signature (c, s_0, s_1) and domain pseudonyms $I_{0,D}, I_{1,D}$ is executed as follows:

- $Q' := (PK_{ICC})^c \cdot g^{s_0} \cdot (PK_M)^{s_1},$
- $A'_0 := (PK_D^{\text{sector}})^{s_0} \cdot (I_0)^c,$
- $A'_1 := (PK_D^{\text{sector}})^{s_1} \cdot (I_1)^c,$
- check whether
 $c \stackrel{?}{=} \text{Hash}(Q', I_{0,D}, A'_0, I_{1,D}, A'_1, PK_D^{\text{sector}}, m).$

In fact, the signature is a non-interactive proof of possession of x_0 and x_1 such that

$$SK_{ICC} = x_0 + x_1 \cdot SK_M \bmod q.$$

Deanonymization of a person holding pseudonyms $I_{0,D}, I_{1,D}$, when $PK_d^{\text{sector}} = g^d$, is based on deriving his public keys $P_0 = I_{0,D}^{1/d}$ and $P_1 = I_{1,D}^{1/d}$.

There are a few serious disadvantages of this scheme. First, the secret keys must be stored securely. Indeed, an adversary holding two pairs of private keys may derive SK_{ICC} and SK_M by solving a system of two linear equations. Another problem is that the Issuer knows the private keys of the user. This can be patched via a procedure from Kutyłowski et al. (2016); however, in that case further changes are required to enable deanonymization.

Pairing-Based Solutions

Pairings enable to provide schemes where the following fundamental conditions are fulfilled:

- seclusiveness,
- lack of whitelists, certificates per domain, etc.,
- the private key of a participant is not known to the Issuer.

The main disadvantage is using pairing groups, which still are not widely deployed in practice.

A pairing $e : G_1 \times G_2 \rightarrow G_T$ is a mapping such that $e(a^s, b^r) = e(a, b)^{s \cdot r}$ for any $a \in G_1, b \in G_2, s, r \in \mathbb{Z}$ and where $e(g_1, g_2) \neq 1$, for generators g_1, g_2 of, respectively, G_1 and G_2 .

The first scheme of this kind has been shown in Bringer et al. (2014). A later example is Hanzlik et al. (2016). A short outline of the first solution is provided below.

The scheme is defined for groups G_1, G_2, G_T of a prime order p , a pairing $e : G_1 \times G_2 \rightarrow G_T$, and generators $g_1, h, u \in G_1, g_2 \in G_2$. The system is based on a secret key y and the corresponding public key $(y_1, y_2) = (h^y, g_2^y)$.

The domain public key PK_D^{sector} is determined as g_1^r , where the secret r is chosen at random.

To join the system, a person must execute an interactive procedure which yields their private key (f, A, x) such that

$$A = (u \cdot h^f)^{1/(x+y)}.$$

First, the person computes $C = h^{f'}$ for some random f' and proposes C to the Issuer. Next, the Issuer chooses f'' and x at random, computes $A := (u \cdot C \cdot h^{f''})^{1/(x+y)}$, and returns (f'', A, x) . Finally, the person computes $f := f' + f'' \bmod p$.

The secret key is used to calculate the pseudonym given domain's public key:

$$N = h^f \cdot (PK_D^{\text{sector}})^x.$$

The signing procedure is based on multiple Schnorr signatures computed in parallel. For a message m , domain public key D , and domain pseudonym N , the following steps are executed:

1. choose $a, r_f, r_x, r_a, r_b, r_d < p$ at random,
2. $T := A \cdot h^a,$
3. $R_1 := h^{r_f} \cdot D^{r_x},$
 $R_2 := N^{r_a} \cdot h^{r_d} \cdot D^{r_b},$
 $R_3 := e(T^{r_x} \cdot h^{r_f r_b} \cdot (y_1)^{r_a}, g_2),$
4. $c := \text{Hash}(D, N, T, R_1, R_2, R_3, m),$
5. $s_x := r_x + c \cdot x \bmod p,$

$$\begin{aligned}s_f &:= r_f + c \cdot f \pmod{p}, \\ s_a &:= r_a + c \cdot a \pmod{p}, \\ s_b &:= r_b + c \cdot a \cdot x \pmod{p}, \\ s_d &:= r_d + c \cdot a \cdot f \pmod{p}.\end{aligned}$$

The signature is a tuple

$$\sigma(m) = (T, c, s_x, s_f, s_a, s_b, s_d).$$

As for Schnorr signatures, verification is based on reconstruction of the hash arguments R_1 , R_2 , and R_3 :

$$\begin{aligned}R'_1 &:= h^{s_f} \cdot D^{s_x} \cdot N^c \\ R'_2 &:= N^{s_a} \cdot h^{s_d} \cdot D^{s_b} \\ R'_3 &:= e(T^{s_x} \cdot h^{s_f s_b} \cdot u^c, G2) \cdot e(h^{s_a} \cdot T^c, y_2).\end{aligned}$$

In the last step, it is checked whether $c = \text{Hash}(D, N, T, R'_1, R'_2, R'_3, m)$.

Lifecycle Issues

Revocation

In the case of losing control over the key by a person, his pseudonyms and signatures are not trustworthy anymore. In order to deal with this problem, a revocation scheme is needed, which invalidates the pseudonyms and the corresponding signatures.

A standard approach in such cases are blacklists (separate for each domain). However, this method may effectively compromise the unlinkability property: when at some moment a single pseudonym in each domain is invalidated, then it is clear that these pseudonyms correspond to the same person. On the other hand, revocation should be done without undue delay, so we cannot wait with publication until a sufficient number of pseudonyms get revoked.

In order to alleviate this problem, one can use cryptographic accumulators known as anonymous credentials. Another, simpler, option is to use Bloom filters: in order to insert a revoked pseudonym $dnym$ in the Bloom filter with sequence number i , we inserts 1's in the filter on positions:

$$\text{Hash}(1, dnym, i), \dots, \text{Hash}(k, dnym, i),$$

where k is chosen so that false positives have sufficiently low probability.

Transition to New Keys

Any anonymization scheme must take into account the fact that a key may get lost or the device implementing it may break down due to hardware aging. Nevertheless, a person must not lose his domain identities.

This problem has been studied in Baldimtsi et al. (2015) in the context of anonymous credentials, and a solution based on commuting signatures Fuchsbauer (2011) was proposed. The problem with domain pseudonyms is that changing the private key would lead to the change of pseudonyms and thereby to re-pseudonymization, which may be a very demanding process for big data.

A general solution to this problem is to export the private key in a permanent form (e.g., printed as a 2D code on a piece of paper) and store it in a private vault. The exported key should be additionally encrypted to safeguard against illegitimate use.

Such encryption and adding a third party to the scheme can be employed to prevent uncontrolled cloning of the devices creating domain signatures, i.e., implementing the same secret key into more than one device. The following procedure may be applied:

- during export, the key is encrypted with the conditional digital encryption scheme Klonowski et al. (2005); the encryption has to be conditioned with a signature of an ID Transition Authority,
- after receiving an authenticating request, the ID Transition Authority verifies its validity and creates a conditional digital signature enabling decryption of the exported key. The signature can be encapsulated by encrypting it with the public key of the user's new device.
- The user uploads the encrypted key and the (encrypted) signature to the new device, and if the device is capable of decrypting the signature, the key may be installed.

Summary

The signature schemes based on pseudonyms related to specific domains of application are a feasible means for providing the level of anonymity

required by law. Their introduction is supported by considerable research in that area, and the already deployed schemes provide good proving grounds for their further improvements.

References

- Baldimtsi F, Camenisch J, Hanzlik L, Krenn S, Lehmann A, Neven G (2015) Recovering lost device-bound credentials. In: Malkin T, Kolesnikov V, Lewko AB, Polychronakis M (eds) Applied cryptography and network security – 13th international conference, ACNS 2015, 2–5 June 2015, Revised selected papers. Lecture notes in computer science, vol 9092. Springer, New York, pp 307–327. https://doi.org/10.1007/978-3-319-28166-7_15
- Bender J, Dagdelen Ö, Fischlin M, Kügler D (2012) Domain-specific pseudonymous signatures for the german identity card. In: Gollmann D, Freiling FC (eds) Proceedings of the 15th international conference on information security, ISC, 19–21 Sept 2012. Lecture notes in computer science, vol 7483. Springer, Passau, pp 104–119. https://doi.org/10.1007/978-3-642-33383-5_7
- Bringer J, Chabanne H, Lescuyer R, Patey A (2014) Efficient and strongly secure dynamic domain-specific pseudonymous signatures for ID documents. In: Christin N, Safavi-Naini R (eds) 18th international conference on financial cryptography and data security, FC 3–7 Mar 2014, Revised selected papers. Lecture notes in computer science, vol 8437. Springer, Christ Church, Barbados, pp 255–272. https://doi.org/10.1007/978-3-662-45472-5_16
- BSI (2016) Technical guideline tr-03110 v2.21 – advanced security mechanisms for machine readable travel documents and eIDAS token. Available at: https://www.bsi.bund.de/EN/Publications/Technical_Guidelines/TR03110/BSITR03110.html
- Fuchsbauer G (2011) Commuting signatures and verifiable encryption. In: Paterson KG (ed) Advances in cryptology – EUROCRYPT 2011–30th annual international conference on the theory and applications of cryptographic techniques, 15–19 May 2011. Proceedings. Lecture notes in computer science, vol 6632. Springer, Tallinn, pp 224–245. https://doi.org/10.1007/978-3-642-20465-4_14
- Hanzlik L, Klucznik K, Kutyłowski M, Dolev S (2016) Local self-organization with strong privacy protection. In: 2016 IEEE Trustcom/BigDataSE/ISPA, 23–26 Aug 2016. IEEE, Tianjin, pp 775–782. <https://doi.org/10.1109/TrustCom.2016.0138>
- Klonowski M, Kutyłowski M, Lauks A, Zagórski F (2005) Conditional digital signatures. In: Katsikas SK, Lopez J, Pernul G (eds) Trust, privacy and security in digital business: second international conference, TrustBus 2005, 22–26 Aug 2005. Proceedings. Lecture notes in computer science, vol 3592. Springer, Copenhagen, pp 206–215. https://doi.org/10.1007/11537878_21
- Kutyłowski M, Krzywiecki Ł, Kubiak P, Koza M (2011) Restricted identification scheme and Diffie-Hellman linking problem. In: Chen L, Yung M, Zhu L (eds) Trusted systems – third international conference, IN-TRUST, 27–29 Nov 2011, Revised selected papers. Lecture notes in computer science, vol 7222. Springer, Beijing, pp 221–238. https://doi.org/10.1007/978-3-642-32298-3_15
- Kutyłowski M, Hanzlik L, Klucznik K (2016) Pseudonymous signature on eIDAS token – implementation based privacy threats. In: Liu JK, Steinfeld R (eds) Information security and privacy – 21st Australasian conference, ACISP, 4–6 July 2016. Proceedings, Part II. Lecture notes in computer science, vol 9723. Springer, Melbourne, pp 467–477. https://doi.org/10.1007/978-3-319-40367-0_31
- Pedersen TP (1991) Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum J (ed) Advances in cryptology – CRYPTO '91, 11th annual international cryptology conference, 11–15 Aug 1991. Proceedings. Lecture notes in computer science, vol 576. Springer, Santa Barbara, pp 129–140. https://doi.org/10.1007/3-540-46766-1_9
- The European Parliament and the Council of the European Union (2016) Regulation (EU) 2016/679 of the European parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (General Data Protection Regulation). Official Journal of the European Union 119(1)

Privacy-Preserving Data Analytics

Do Le Quoc¹, Martin Beck¹, Pramod Bhatotia², Ruichuan Chen^{3,4}, Christof Fetzer¹, and Thorsten Strufe¹

¹TU Dresden, Dresden, Germany

²The University of Edinburgh and Alan Turing Institute, Edinburgh, UK

³Nokia Bell Labs, Stuttgart, Germany

⁴Nokia Bell Labs, NJ, USA

Introduction

Nowadays, many modern online services continuously collect users' private data for real-time analytics. This data normally arrives as a data

stream and in huge volumes, requiring real-time stream processing based on distributed systems (Carbone et al. 2015; Apache Spark Streaming).

In the current ecosystem of data analytics, the analysts usually have direct access to users' private data and must be trusted not to abuse it. However, this trust has been violated in the past ([HealthCare.gov Sends Personal Data to Dozens of Tracking Websites; SEC Charges Two Employees of a Credit Card Company with Insider Trading](#)). A pragmatic ecosystem has two desirable but contradictory design requirements: (i) stronger privacy guarantees for users and (ii) high-utility stream analytics in real time. Users seek stronger privacy, while analysts strive for high-utility analytics in real time.

To meet these two design requirements, there is a surge of novel computing paradigms that address these concerns, albeit *separately*. Two such paradigms are *privacy-preserving analytics* to protect user privacy and *approximate computation* for real-time analytics.

Privacy-preserving analytics. Recent privacy-preserving analytics systems favor a distributed architecture to avoid central trust (see section “[Related Work](#)” for details), where users’ private data is stored locally on their respective client devices. Data analysts use a publish-subscribe mechanism to perform aggregate queries over the distributed private dataset of a large number of clients. Thereafter, such systems add noise to the aggregate output to provide useful privacy guarantees, such as differential privacy (Dwork 2006). Unfortunately, these state-of-the-art systems normally deal with single-shot batch queries, and, therefore, these systems cannot be used for real-time stream analytics.

Approximate computing. Approximate computation is based on the observation that many data analytics jobs are amenable to an approximate rather than the exact output (see section “[Related Work](#)” for details). Such applications include speech recognition, computer vision, machine learning, and recommender systems. For such an approximate

workflow, it is possible to trade accuracy by computing over a subset (usually selected via a sampling mechanism) instead of the entire input dataset. Thereby, data analytics systems based on approximate computation can achieve low latency and efficient utilization of resources. However, the existing systems for approximate computation assume a centralized dataset, where the desired sampling mechanism can be employed. Thus, existing systems are not compatible with the distributed privacy-preserving analytics systems.

Combining privacy-preserving analytics and approximate computing. This paper makes the observation that the two computing paradigms, i.e., privacy-preserving analytics and approximate computation, are complementary. Both paradigms strive for an approximate instead of the exact output, but they differ in their *means* and *goals* for approximation. Privacy-preserving analytics adds explicit *noise* to the aggregate query output to protect user privacy, whereas approximate computation relies on a representative *sampling* of the entire dataset to compute over only a subset of data items to enable low-latency/efficient analytics. Therefore, this work combines these two existing paradigms together in order to leverage the benefits of both. The high-level idea is to achieve privacy (via approximation) by directly computing over a subset of sampled data items (instead of computing over the entire dataset) and then adding an explicit noise for privacy preservation.

To realize this combination, this paper presents the design of an approximation mechanism that also achieves privacy-preserving goals for stream analytics. This design targets a distributed setting, where users’ private data is stored locally on their respective personal devices, and an analyst issues a streaming query for analytics over the distributed private dataset of users. The analyst’s streaming query is executed on the users’ data periodically (a configurable epoch), and the query results are transmitted to a centralized aggregator via a set of proxies. The analyst interfaces with the aggregator to get the aggregate query output periodically.

Two core techniques are employed to achieve the goal. Firstly, *sampling* (Moore 1999) is directly employed at the user site for approximate computation, where each user randomly decides whether to participate in answering the query in the current epoch. Since the sampling is employed at the data source, instead of sampling at a centralized infrastructure, the proposed approach can squeeze out the desired data size (by controlling the sampling parameter) from the very first stage in the analytics pipeline, which is essential in low-latency environments.

Secondly, if the user participates in the query answering process, a *randomized response* (Fox and Tracy 1986) mechanism is performed to add noise to the query output at the user site, again locally at the source of the data in a decentralized fashion. In particular, each user locally randomizes the truthful answer to the query to achieve the differential privacy guarantees (section “[Step II: Answering Queries at Clients](#)”). Since the noise is added at the source of data, instead of adding the explicit noise to the aggregate output at a trusted aggregator or proxies, the proposed approach enables a truly “synchronization-free” distributed architecture, which requires *no coordination* among proxies and the aggregator for the mandated noise addition.

The last, but not the least, silver bullet of the design: it turns out that the combination of the two aforementioned techniques (i.e., sampling and randomized response) leads us to achieve zero-knowledge privacy (Gehrke et al. 2011), a privacy bound tighter than the state-of-the-art differential privacy (Dwork 2006).

To summarize, this paper presents the design of PRIVAPPROX— a practical system for privacy-preserving stream analytics in real time. In particular, the system is a novel combination of the sampling and randomized response techniques, as well as a scalable “synchronization-free” routing scheme which employs a light-weight XOR-based encryption scheme (Chen et al. 2013). The resulting system ensures zero-knowledge privacy, anonymization, and unlinkability for users (section “[System Model](#)”).

Overview

System Architecture

PRIVAPPROX is designed for privacy-preserving stream analytics on distributed users’ private dataset. This system consists of four main components: clients, proxies, aggregator, and analysts.

Clients locally store users’ private data on their respective personal devices and subscribe to queries from the system. *Analysts* publish streaming queries to the system and also specify a query execution budget. The query execution budget can either be in the form of latency guarantees/S-LAs, output quality/accuracy, or the computing resources for query processing. PRIVAPPROX ensures that the computation remains within the specified budget.

At a high-level, the system works as follows: a query published by an analyst is distributed to clients via the aggregator and proxies. Clients answer the analyst’s query locally over the users’ private data using a privacy-preserving mechanism. Client answers are transmitted to the aggregator via anonymizing *proxies*. The *aggregator* aggregates received answers from the clients to provide privacy-preserving stream analytics to the analyst.

System Model

Query model. PRIVAPPROX supports the SQL query language for *analysts* to formulate streaming queries, which are executed periodically at the clients as sliding window computations (Bhatotia et al. 2012a). While queries can be complex, the results of a query are expressed as counts within histogram buckets, i.e., each bucket represents a range of the query’s answer values. Specifically, each query answer is represented in the form of binary buckets, where each bucket stores a value “1” or “0” depending on whether or not the answer falls into the value range represented by that bucket. For example, an analyst can learn the driving speed distribution across all vehicles in San Francisco

by formulating an SQL query “SELECT speed FROM vehicle WHERE location=‘San Francisco’.” The analyst can then define 12 answer buckets on speed: “0,” “1~10,” “11~20,” …, “81~90,” “91~100,” and “> 100.” If a vehicle is moving at 15 mph in San Francisco, it answers “1” for the third bucket and “0” for all others.

The query model supports not only numeric queries as described above but also non-numeric queries. For non-numeric queries, each bucket is specified by a matching rule or a regular expression.

Threat model. Analysts are potentially malicious. They may try to violate the PRIVAPPROX’s privacy model (described later), i.e., de-anonymize clients, build profiles through the linkage of queries and answers, or remove the added noise from answers.

Clients are potentially malicious. They could generate false or invalid responses to distort the query result for the analyst. However, the proposed system does not defend against the Sybil attack (Douceur 2002), which is beyond the scope of this work (Wang et al. 2016a).

Proxies are also potentially malicious. They may transmit messages between clients and the aggregator in contravention of the system protocols. PRIVAPPROX includes at least two proxies, and there are at least two proxies which do not collude with each other. The *aggregator* is assumed to be honest but curious. The aggregator faithfully conforms to the system protocols but may try to exploit the information about clients. The aggregator does not collude with any proxy nor the analyst.

Privacy model. PRIVAPPROX provides three important privacy properties: (i) zero-knowledge privacy, (ii) anonymity, and (iii) unlinkability.

All aggregate query results in the system are independently produced under the *zero-knowledge privacy* guarantees (Gehrke et al. 2011). The zero-knowledge privacy metric builds upon differential privacy (Dwork 2006) and

provides a tighter bound on privacy guarantees compared to differential privacy. Informally, zero-knowledge privacy states that essentially everything that an adversary can learn from the output of an zero-knowledge private mechanism could also be learned using the aggregate information. *Anonymity* means that no system component can associate query answers or query requests with a specific client. Finally, *unlinkability* means that no system component can join any pair of query requests or answers to the same client, even to the same anonymous client. The formal definition, analysis, and proof are described in the technical report (Quoc et al. 2017a).

Design

PRIVAPPROX consists of two main phases: *submitting queries* and *answering queries*. In the first phase, an analyst submits a query (along with the execution budget) to clients via the aggregator and proxies. In the second phase, the query is answered by the clients in the reverse direction.

Submitting Queries

To perform statistical analysis over users’ private data streams, an analyst creates a query using the query model described in section “[System Model](#)”. In particular, each query consists of the following fields and is signed by the analyst for non-repudiation:

$$\text{Query} := \langle Q_{ID}, SQL, A[n], f, w, \delta \rangle \quad (1)$$

- Q_{ID} denotes a unique identifier of the query. This can be generated by concatenating the identifier of the analyst with a serial number unique to the analyst.
- SQL denotes the actual SQL query, which is passed on to clients and executed on their respective personal data.
- $A[n]$ denotes the format of a client’s answer to the query. The answer is an n -bit vector where

each bit associates with a possible answer value in the form of a “0” or “1” per index (or answer value range).

- f denotes the answer frequency, i.e., how often the query needs to be executed at clients.
- w denotes the window length for sliding window computations (Bhatotia et al. 2014). For example, an analyst may only want to aggregate query results for the last ten minutes, which means the window length is ten minutes.
- δ denotes the sliding interval for sliding window computations. For example, an analyst may want to update the query results every one minute, and so the sliding interval is set to one minute.

After forming the query, the analyst sends the query, along with the query execution budget, to the aggregator. Once receiving the pair of the query and query budget from the analyst, the aggregator first converts the query budget into system parameters for sampling and randomization. The system operator can set the sampling fraction using resource prediction model (Wieder et al. 2010a, b, 2012) for any given SLA. Hereafter, the aggregator forwards the query and the converted system parameters to clients via proxies.

Answering Queries

After receiving the query and system parameters, the query is answered by clients and processed by the system to produce the result for the analyst. The query answering process involves four steps including (i) sampling at clients for low-latency approximation, (ii) randomizing answers for privacy preservation, (iii) transmitting answers via proxies for anonymization and unlinkability, and finally, (iv) aggregating answers with error estimation to give a confidence level on the approximate result. (The detailed algorithms are covered in Quoc et al. (2017a, b).)

Step I: Sampling at Clients

PRIVAPPROX makes use of approximate computation to achieve low-latency execution by computing over a subset of data items instead of the entire input dataset. Specifically, the system

builds on sampling-based techniques (Krishnan et al. 2016; Quoc et al. 2017c, d). To keep the private data stored at individual clients, PRIVAPPROX applies an input data sampling mechanism locally at the clients. In particular, the system uses *simple random sampling* (SRS) (Moore 1999).

Note that, this work assumes that all clients produce the input stream with data items following the same distribution, i.e., all clients’ data streams belong to the same stratum. The sampling mechanism can be further extended with the *stratified sampling* technique (Krishnan et al. 2016; Quoc et al. 2017c, d) to deal with varying distributions of data streams. The algorithm and evaluation of stratified sampling are covered in the technical report (Quoc et al. 2017a).

Step II: Answering Queries at Clients

Clients that participate in the query answering process make use of the *randomized response* technique (Fox and Tracy 1986) to preserve answer privacy, with *no* synchronization among clients. Randomized response works as follows: suppose an analyst sends a query to individuals to obtain the statistical result about a sensitive property. To answer the query, a client locally randomizes its answer to the query (Fox and Tracy 1986). Specifically, the client flips a coin, if it comes up heads, then the client responds its truthful answer; otherwise, the client flips a second coin and responds “Yes” if it comes up heads or “No” if it comes up tails. The privacy is preserved via the ability to refuse responding truthful answers (see details in Quoc et al. (2017a, b)).

It is worth mentioning that, combining the randomized response with the sampling technique described in Step I, PRIVAPPROX achieves not only differential privacy but also zero-knowledge privacy (Gehrke et al. 2011) which is a privacy bound tighter than differential privacy. The detailed proof is described in the technical report (Quoc et al., 2017a).

Step III: Transmitting Answers via Proxies

After producing randomized responses, clients transmit them to the aggregator via the prox-

ies. To achieve anonymity and unlinkability of the clients against the aggregator and analysts, PRIVAPPROX utilizes the XOR-based encryption together with source rewriting, which has been used for anonymous communications (Chen et al. 2013; Dingledine et al. 2004). The underlying idea of this encryption is simple: if Alice wants to send a message M of length l to Bob, then Alice and Bob share a secret M_K (in the form of a random bit-string of length l). To transmit the message M privately, Alice sends an encrypted message " $M_E = M \oplus M_K$ " to Bob, where " \oplus " denotes the bit-wise XOR operation. To decrypt the message, Bob again uses the bit-wise XOR operation: $M = M_E \oplus M_K$ (see details in Quoc et al. 2017a, b).

Step IV: Generating Result at the Aggregator

At the aggregator, all data streams ((M_{ID}, M_E) and (M_{ID}, M_{K_i})) are received and can be joined together to obtain a unified data stream. Specifically, the associated M_E and M_{K_i} are paired by using the message identifier M_{ID} . To decrypt the original randomized message M from the client, the XOR operation is performed over M_E and M_K : $M = M_E \oplus M_K$ with M_K being the XOR of all M_{K_i} : $M_K = \bigoplus_{i=2}^n M_{K_i}$. As the aggregator cannot identify which of the received messages is M_E , it just XORs all the n received messages to decrypt M .

Note that an adversarial client might answer a query many times in an attempt to distort the query result. However, this problem can be handled, for example, by applying the *triple splitting* technique (Chen et al. 2013).

Error bound estimation. PRIVAPPROX provides an error bound estimation for the aggregate query results. The accuracy loss in PRIVAPPROX is caused by two processes: (i) sampling and (ii) randomized response. Since the accuracy loss of these two processes is statistically independent (see details in Quoc et al. (2017b, a)), PRIVAPPROX estimates the accuracy loss of each process separately. In addition, independent of the error induced by randomized response, the error coming from sampling is simply being added upon. Following

this, the system sums up both independently estimated errors to provide the total error bound of the query results. To estimate the accuracy loss of the randomized response process, PRIVAPPROX makes use of an experimental method. It performs several micro-benchmarks at the beginning of the query answering process (without performing the sampling process) to estimate the accuracy loss caused by randomized response. On the other hand, to estimate the accuracy loss of the sampling process, PRIVAPPROX applies the statistical theory of the sampling techniques (see details in Quoc et al. (2017b, a)).

Related Work

Privacy-preserving analytics. Since the notion of differential privacy (Dwork 2006; Dwork et al. 2006b), a plethora of systems have been proposed to provide differential privacy with centralized databases (McSherry and Mahajan 2010; Mohan et al. 2012). In practice, however, such central trust can be abused, leaked, or subpoenaed (HealthCare.gov Sends Personal Data to Dozens of Tracking Websites; SEC Charges Two Employees of a Credit Card Company with Insider Trading).

To overcome the limitations of the centralized database schemes, recently a flurry of systems have been proposed with a focus on preserving user privacy (mostly, differential privacy) in a distributed setting where the private data is kept locally (Guha et al. 2011; Chen et al. 2013; Dwork et al. 2006a). However, these systems are designed to deal with the "one-shot" batch queries only, whereby the data is assumed to be static.

To overcome the limitations of the aforementioned systems, several differentially private stream analytics systems have been proposed (Dwork et al. 2010; Chan et al. 2011, 2012; Shi et al. 2011; Rastogi and Nath 2010). Unfortunately, these systems still contain several technical shortcomings that limit their practicality. One of the first systems (Dwork et al. 2010) updates the query result only if the user's private data changes significantly and does

not support stream analytics over an unlimited time period. Subsequent systems (Chan et al. 2011; Hubert Chan et al. 2012) remove the limit on the time period but introduce extra system overheads. Some systems (Shi et al. 2011; Rastogi and Nath 2010) leverage expensive secret sharing cryptographic operations to produce noisy aggregate query results. These systems, however, cannot work at large scale under churn; moreover, in these systems, even a single malicious user can substantially distort the aggregate results without detection. Recently, some other privacy-preserving distributed stream monitoring systems have been proposed (Chan et al. 2012). However, they all require some form of synchronization and are tailored for heavy-hitter monitoring only. Streaming data publishing systems like Wang et al. (2016b) use a stream-privacy metric at the cost of relying on a trusted party to add noise. In contrast, PRIVAPPROX does not require a trusted proxy or aggregator to add noise. Furthermore, PRIVAPPROX provides stronger privacy properties (i.e., zero-knowledge privacy).

Sampling and randomized response. Sampling and randomized response, also known as input perturbation techniques, are being studied in the context of privacy-preserving analytics, albeit they are explored separately. For instance, the relationship between sampling and privacy is being investigated to provide k-anonymity (Chaudhuri and Mishra 2006), differential privacy (Mohan et al. 2012), and crowd-blending privacy (Gehrke et al. 2012). In contrast, this paper shows that sampling combined with randomized response achieves the zero-knowledge privacy, a privacy bound strictly stronger than the state-of-the-art differential privacy.

Approximate computation. Approximation techniques such as sampling (Al-Kateb and Lee 2010), sketches (Cormode et al. 2012), and online aggregation (Hellerstein et al. 1997) have been well-studied over the decades in the databases community. Recently, sampling-based systems (Quoc et al. 2017c, d; Krishnan et al. 2016) have also been shown effective for “Big

Data” analytics. In particular, our work builds on IncApprox (Krishnan et al. 2016), a data analytics system that combines incremental computation (Bhatotia et al. 2011a, b, 2012b, 2015; Bhatotia 2015) and approximate computation. However, PRIVAPPROX differs in two main aspects. First, the system performs sampling in a distributed way as opposed to sampling in a centralized dataset. Second, PRIVAPPROX extends sampling with randomized response for privacy-preserving analytics.

Conclusion

This paper presents PRIVAPPROX, a privacy-preserving stream analytics system. The approach in PRIVAPPROX builds on the observation that both computing paradigms – privacy-preserving data analytics and approximate computation – strive for approximation and can be combined together to leverage the benefits of both.

Cross-References

► Scalable Big Data Privacy with MapReduce

References

- Al-Kateb M, Lee BS (2010) Stratified reservoir sampling over heterogeneous data streams. In: Proceedings of the 22nd international conference on scientific and statistical database management (SSDBM)
- Apache spark streaming. <http://spark.apache.org/streaming>. Accessed Nov 2017
- Bhatotia P (2015) Incremental parallel and distributed systems. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS)
- Bhatotia P, Wieder A, Akkus IE, Rodrigues R, Acar UA (2011a) Large-scale incremental data processing with change propagation. In: Proceedings of the conference on hot topics in cloud computing (HotCloud)
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquini R (2011b) Incoop: MapReduce for incremental computations. In: Proceedings of the ACM symposium on cloud computing (SoCC)
- Bhatotia P, Dischinger M, Rodrigues R, Acar UA (2012a) Slider: incremental sliding-window computations for large-scale data analysis. Technical Report

- MPI-SWS-2012-004, MPI-SWS. <http://www.mpi-sws.org/tr/2012-004.pdf>
- Bhatotia P, Rodrigues R, Verma A (2012b) Shredder: GPU-accelerated incremental storage and computation. In: Proceedings of USENIX conference on file and storage technologies (FAST)
- Bhatotia P, Acar UA, Junqueira FP, Rodrigues R (2014) Slider: incremental sliding window analytics. In: Proceedings of the 15th international middleware conference (Middleware)
- Bhatotia P, Fonseca P, Acar UA, Brandenburg B, Rodrigues R (2015) iThreads: a threading library for parallel incremental computation. In: Proceedings of the 20th international conference on architectural support for programming languages and operating systems (ASPLOS)
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink: stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Committee Data Eng 36(4)
- Chan THH, Shi E, Song D (2011) Private and continual release of statistics. ACM Trans Inf Syst Secur 14(3), 26
- Chan THH, Li M, Shi E, Xu W (2012) Differentially private continual monitoring of heavy hitters from distributed streams. In: Proceedings of the 12th international conference on privacy enhancing technologies (PETS)
- Chaudhuri K, Mishra N (2006) When random sampling preserves privacy. In: Proceedings of the 26th annual international conference on advances in cryptology (CRYPTO)
- Chen R, Akkus IE, Francis P (2013) SplitX: high-performance private analytics. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM)
- Cormode G, Garofalakis M, Haas PJ, Jermaine C (2012) Synopses for massive data: samples, histograms, wavelets, sketches. Found Trends Databases 4(1–3):1–294
- Dingledine R, Mathewson N, Syverson P (2004) Tor: the second-generation onion router. Technical report, DTIC Document
- Douceur JR (2002) The Sybil attack. In: Proceedings of 1st international workshop on peer-to-peer systems (IPTPS)
- Dwork C (2006) Differential privacy. In: Proceedings of the 33rd international colloquium on automata, languages and programming, part II (ICALP)
- Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006a) Our data, ourselves: privacy via distributed noise generation. In: Proceedings of the 24th annual international conference on the theory and applications of cryptographic techniques (EUROCRYPT)
- Dwork C, McSherry F, Nissim K, Smith A (2006b) Calibrating noise to sensitivity in private data analysis. In: Proceedings of the third conference on theory of cryptography (TCC)
- Dwork C, Naor M, Pitassi T, Rothblum GN (2010) Differential privacy under continual observation. In: Proceedings of the ACM symposium on theory of computing (STOC)
- Fox JA, Tracy PE (1986) Randomized response: a method for sensitive surveys. Sage Publications, Beverly Hills
- Gehrke J, Lui E, Pass R (2011) Towards privacy for social networks: a zero-knowledge based Definitions of privacy. In: Theory of cryptography
- Gehrke J, Hay M, Lui E, Pass R (2012) Crowd-blending privacy. In: Proceedings of the 32nd annual international conference on advances in cryptology (CRYPTO)
- Guha S, Cheng B, Francis P (2011) Privad: practical privacy in online advertising. In: Proceedings of the 8th symposium on networked systems design and implementation (NSDI)
- Hellerstein JM, Haas PJ, Wang HJ (1997) Online aggregation. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)
- HealthCare.gov sends personal data to dozens of tracking websites. <https://www.eff.org/deeplinks/2015/01/healthcare.gov-sends-personal-data>. Accessed Nov 2017
- Hubert Chan Th, Shi E, Song D (2012) Privacy-preserving stream aggregation with fault tolerance. In: Proceedings of 16th international conference on financial cryptography and data security (FC)
- Krishnan DR, Quoc DL, Bhatotia P, Fetzer C, Rodrigues R (2016) IncApprox: a data analytics system for incremental approximate computing. In: Proceedings of the 25th international conference on world wide web (WWW)
- McSherry F, Mahajan R (2010) Differentially-private network trace analysis. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM)
- Mohan P, Thakurta A, Shi E, Song D, Culler D (2012) GUPT: privacy preserving data analysis made easy. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data (SIGMOD)
- Moore DS (1999) The basic practice of statistics, 2nd edn. W. H. Freeman & Co., New York
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017a) Privacy preserving stream analytics: the marriage of randomized response and approximate computing. <https://arxiv.org/abs/1701.05403>
- Quoc DL, Beck M, Bhatotia P, Chen R, Fetzer C, Strufe T (2017b) PrivApprox: privacy-preserving stream analytics. In: Proceedings of the 2017 USENIX conference on USENIX annual technical conference (USENIX ATC)
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017c) Approximate stream analytics in Apache Flink and Apache Spark streaming. CorR, abs/1709.02946
- Quoc DL, Chen R, Bhatotia P, Fetzer C, Hilt V, Strufe T (2017d) StreamApprox: approximate computing for stream analytics. In: Proceedings of the international middleware conference (Middleware)
- Rastogi V, Nath S (2010) Differentially private aggregation of distributed time-series with transformation and encryption. In: Proceedings of the international conference on management of data (SIGMOD)

- SEC Charges Two Employees of a Credit Card Company with Insider Trading. <http://www.sec.gov/litigation/litreleases/2015/lr23179.htm>. Accessed Nov 2017
- Shi E, Chan TH, Rieffel EG, Chow R, Song D (2011) Privacy-preserving aggregation of time-series data. In: Proceedings of the symposium on network and distributed system security (NDSS)
- Wang G, Wang B, Wang T, Nika A, Zheng H, Zhao BY (2016a) Defending against Sybil devices in crowd-sourced mapping services. In: Proceedings of the 14th annual international conference on mobile systems, applications, and services (MobiSys)
- Wang Q, Zhang Y, Lu X, Wang Z, Qin Z, Ren K (2016b) RescuedP: real-time spatio-temporal crowd-sourced data publishing with differential privacy. In: Proceedings of the 35th annual IEEE international conference on computer communications (INFOCOM)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010a) Brief announcement: modelling mapreduce for optimal execution in the cloud. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing (PODC)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2010b) Conductor: orchestrating the clouds. In: Proceedings of the 4th international workshop on large scale distributed systems and middleware (LADIS)
- Wieder A, Bhatotia P, Post A, Rodrigues R (2012) Orchestrating the deployment of computations in the cloud with conductor. In: Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI)

Privacy-Preserving Record Linkage

Dinusha Vatsalan^{1,2}, Dimitrios Karapiperis³, and Vassilios S. Verykios³

¹Data61, CSIRO, Eveleigh, NSW, Australia

²Research School of Computer Science, The Australian National University, Acton, ACT, Australia

³School of Science and Technology, Hellenic Open University, Patras, Greece

Synonyms

Blind data linkage; Private data integration; Private data matching; Private record linkage

Definitions

Given several databases containing person-specific data held by different organizations, privacy-preserving record linkage (PPRL) aims to identify and link records that correspond to the same entity/individual across different databases based on the matching of personal identifying attributes, such as name and address, without revealing the actual values in these attributes due to privacy concerns. The output is a set of matching clusters containing records of the same entity.

Overview

In the current era of Big Data, personal data about people, such as customers, patients, tax payers, and clients, are dispersed in multiple different sources collected by different organizations. Businesses and services have begun to leverage tremendous opportunities and insights provided by linked and integrated data. Examples range from healthcare, businesses, and social sciences to government services and national security. Linking data is also used as a preprocessing step in many data mining and analytics projects in order to clean, enrich, and understand data for quality results (Christen, 2012).

However, the growing concerns of privacy and confidentiality issues about personal data pose serious constraints to share and exchange such data across organizations for linking. Since a unique entity identifier is not available in different data sources, linking of records from different databases needs to rely on available personal identifying attributes, such as names, dates of birth, and addresses. Known as quasi identifiers (QIDs), these values in combination not only allow uniquely identifying individuals but also reveal private and sensitive information about them (Vatsalan et al., 2013).

Privacy-preserving record linkage (PPRL) aims to identify the same entities from different databases by linking records based on encoded and/or encrypted QIDs, such that no sensitive

information of the entities is revealed to any internal (parties involved in the process) and external (external adversaries and eavesdroppers) parties. While a variety of techniques and methodologies have been developed for PPRL over the past two decades, as surveyed in (Vatsalan et al., 2013), this research field is still open to several challenges, especially with the Big Data revolution.

A typical PPRL process involves several steps, starting from preprocessing databases, encoding or encrypting records using the privacy masking functions (Vatsalan et al., 2013), applying blocking or other forms of complexity reduction methods (Christen, 2012), then matching records based on their QIDs using approximate comparison functions (Christen, 2012), and finally clustering or classifying matching records that correspond to the same entity. Additional steps of evaluation of the linkage as well as manual review of certain records are followed in non-PPRL applications. However, these two steps require more research for PPRL as they generally require access to raw data and ground truth which is not possible in a privacy-preserving context.

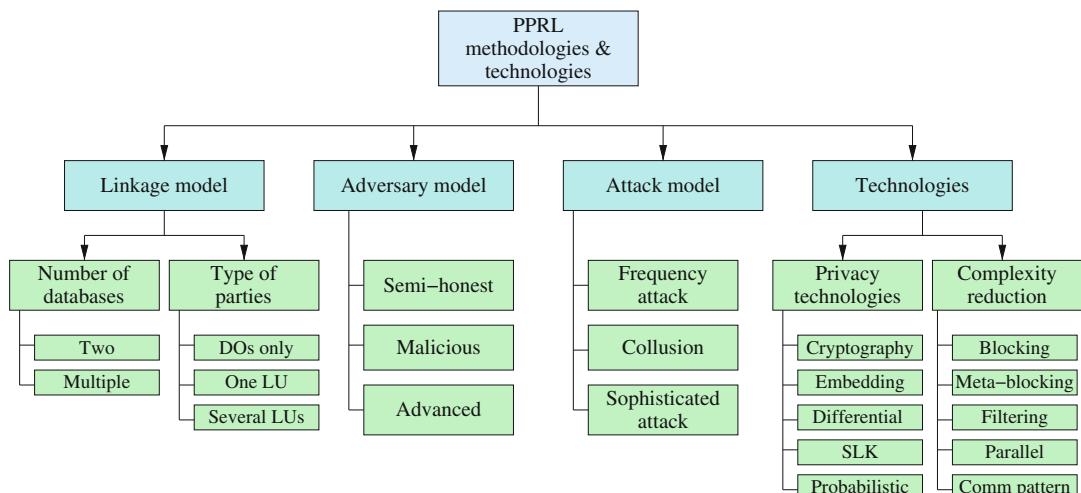
Despite several challenges, the output of PPRL could certainly bring in enormous potential in the Big Data era for businesses, government agencies, and research organizations. PPRL has been an active area of research in the recent

times, and it is now being applied in real-world applications (Boyd et al., 2015; Randall et al., 2014). In this chapter, we will describe some of the key research findings of PPRL and example PPRL applications. We also discuss directions for future research in PPRL.

Key Research Findings

In this section, we present the key research findings in PPRL with regard to the methodologies and technologies used, as characterized in Fig. 1. Considering the number and type of parties involved, the linkage models can be categorized as:

- Two database owners (DOs)** involve in two-party protocols in order to identify the matching records in their databases. This linkage model requires more sophisticated techniques to ensure that the parties do not learn any sensitive information about each other's data. Linking two databases has a naive complexity of quadratic in the size of databases.
- Two DOs with a linkage unit (LU)** participate in three-party protocols where a trusted LU (third party) conducts the linkage. A major drawback of LU-based protocols is that they require a trusted LU in order to avoid possible collusion attacks, as described below.



Privacy-Preserving Record Linkage, Fig. 1 A taxonomy of methodologies and technologies used in PPRL

3. **Two DOs with several LUs** is the model where several LUs are used to link two databases. For example, one is responsible for secret keys management, and another one for facilitating the complexity reduction step, while matching of records is conducted by a different LU. Separating the tasks across several LUs reduces the amount of information learned by a single party; however collusion could compromise the privacy.
4. **Multiple DOs without or with LU(s)** involve in multiparty linkage where the aim is to identify cluster of records from multiple (more than two) databases that refer to the same entity. The naive complexity of linking multiple databases becomes exponential, and the linkage process becomes complicated with the increase of number of databases. Processing can be distributed among multiple parties to improve efficiency as well as privacy by reducing the amount of information learned by a single party.

Different adversary models are assumed in PPRL methodologies (Vatsalan et al., 2013):

1. **Honest-but-curious/semi-honest** model is the most commonly used adversary model in existing PPRL techniques (Vatsalan et al., 2013), where the parties are assumed to honestly follow the steps of the protocol while being curious to learn from the information they received throughout the protocol. This model is not realistic in real applications due to the weak assumption of privacy against adversarial attacks.
2. **Malicious model** provides a strong assumption of privacy by assuming that the parties involved in the protocol may not follow the steps of the protocol in terms of deviating from the protocol, sending false input, and behaving arbitrarily. More complex and advanced privacy techniques are required to make the protocols resistant against such malicious adversaries.
3. **Advanced models** are hybrid models that lie in between the semi-honest model, which is not realistic, and the malicious model, which requires computationally expensive

techniques. Two such advanced models are accountable computing and covert model.

Several attack models have been developed for PPRL techniques to investigate the resistance of such techniques:

1. **Frequency-based attacks** are most commonly used, where the frequency of encoded values is mapped to the frequency of known unencoded values (Vatsalan et al., 2014).
2. **Collusion** attacks are possible in LU-based and multiparty models where subsets of parties collude to learn another party's data (Vatsalan et al., 2014).
3. **More sophisticated attacks** have been developed against certain privacy techniques. For example, Bloom filters (as described below) are susceptible to cryptanalysis attacks, which allow the iterative mapping of hash-encoded values back to their original values depending upon the parameter settings (Christen et al., 2017; Kuzu et al., 2011).

With regard to privacy technologies, a variety of techniques has been developed for PPRL. We categorize the key findings as:

1. **Cryptography** refers to secure multiparty computation techniques, such as homomorphic encryptions, secret sharing, and secure vector operations (Lindell and Pinkas, 2009). These techniques are provably secure and highly accurate; however, they are computationally expensive. An example PPRL technique based on cryptographic techniques is the secure edit distance algorithm for matching two strings or genome sequences (Atallah et al., 2003), which is quadratic in the length of the strings.
2. **Embedding techniques** allow data to be mapped into a multidimensional metric space while preserving the distances between original data. It is difficult to determine the appropriate dimensionality for the metric space. A recent work proposed a framework for embedding string and numerical data with theoretical guarantees for rigorous specifi-

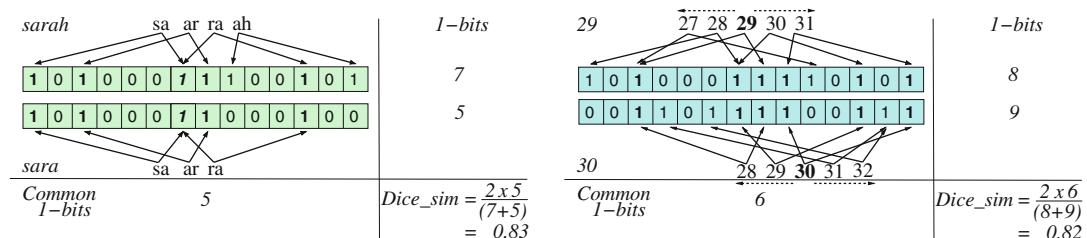
- cation of space dimensionality (Karapiperis et al., 2017).
3. **Differential privacy** is a rigorous definition that provides guarantees of indistinguishability of an individual in the data with high probability. It has been used in PPRL to perturb data by adding noise such that every individual in the dataset is indistinguishable (Vatsalan et al., 2013, 2014). However, adding noise incurs utility loss and volume increase.
 4. **Statistical linkage key (SLK)** contains derived values from QIDs which is generated using a combination of components of a set of QIDs. An example is the SLK-581 consisting of the second and third letters of first name; the second, third, and fifth letters of surname; full date of birth; and sex, which was developed by the Australian Institute of Health and Welfare to link records from the Home and Community Care datasets (Randall et al., 2016). A recent study has shown that SLK-based techniques fall short in providing sufficient privacy protection and sensitivity (Randall et al., 2016).
 5. **Probabilistic methods** are the most widely used techniques for practical PPRL applications due to their efficiency and controllable privacy-utility trade-off. These methods use probabilistic data structures for mapping/encoding data such that the actual distances between original data are preserved depending on the false-positive probability of the mapping. A recent study has shown that, if effectively used, probabilistic techniques can achieve high linkage quality comparable to using unencoded records (Randall et al., 2014).

For example, Bloom filter encoding is one probabilistic method that has been used in several PPRL solutions (Durham, 2012; Randall et al., 2014; Schnell et al., 2009; Sehili et al., 2015; Vatsalan and Christen, 2012, 2014). A Bloom filter b_i is a bit array of length l bits where all bits are initially set to 0. k independent hash functions, h_j , with $1 \leq j \leq k$, are used to map each of the elements s in a set S into the Bloom filter by setting the bit positions $h_j(s)$, $1 \leq j \leq k$ to 1. As shown in Fig. 2 (left), the set S of q -grams (substrings of length q) of QID values of each record can be hash-mapped into a Bloom filter. The resulting Bloom filters can be matched using a similarity function, such as the Dice-coefficient (Christen, 2012):

$$\text{Dice_sim}(b_1, \dots, b_P) = \frac{P \times c}{\sum_{i=1}^P x_i}, \quad (1)$$

where P is the number of Bloom filters compared, c is the number of common bit positions that are set to 1 in all P Bloom filters, and x_i is the number of bit positions set to 1 in b_i , $1 \leq i \leq P$. The matching can either be done by a LU (Durham, 2012; Schnell et al., 2009) or collaboratively by the DOs (Vatsalan and Christen, 2012, 2014). Recently, Bloom filter-based matching of numerical data has been developed, which hash-maps a set S of neighbor values to allow approximate matching (shown in Fig. 2 (right)) (Vatsalan and Christen, 2016).

Counting Bloom filter is another probabilistic method where counts of elements are



Privacy-Preserving Record Linkage, Fig. 2 Bloom filter-based matching for string (Schnell et al., 2009) (left) and numerical (Vatsalan and Christen, 2016) (right) data

stored in an integer array. Counting Bloom filter has been used for PPRL of multiple databases (Vatsalan et al., 2016) where the Bloom filters from different databases are summed to generate a counting Bloom filter which is used by the LU to classify the set of records as matches or not. Counting Bloom filters provide improved privacy compared to Bloom filters as they store only the summary information; however, they incur increased memory cost. Count-min sketch is a memory-efficient probabilistic data structure that can be used to store count information and has been used for linking multiple databases (Karapiperis et al., 2015).

The bottleneck of the PPRL process is the comparison of records among different databases, which is equal to the product of the sizes of the databases. Computational technologies have been used to improve the scalability of PPRL:

1. **Blocking** is defined on selected attributes and it partitions the records in a database into several blocks or clusters such that comparison can be restricted to the records of the same block. A variety of blocking techniques has been developed (Vatsalan et al., 2013). Recent examples are randomized blocking methods based on locality-sensitive hashing, which provide theoretical guarantees for identifying similar record pairs in the embedding space with high probability (Durham, 2012; Karapiperis and Verykios, 2015).
2. **Meta-blocking** is the process of restructuring a collection of generated blocks to be compared in the next step such that unnecessary comparisons are pruned. Block processing for PPRL only received much attention in the recent years with the aim to improve the scalability in Big Data applications by employing such techniques along with blocking techniques (Karakasidis et al., 2015; Ranbaduge et al., 2016).
3. **Filtering** is an optimization for a particular comparison function which utilizes different filtering or thresholding techniques to elimi-

nate pairs/sets of records that cannot meet the similarity threshold for the selected similarity measures (Sehili et al., 2015; Vatsalan and Christen, 2012).

4. **Parallel/distributed processing** for PPRL has only seen limited attention so far. Parallel linkage aims at improving the execution time proportionally to the number of processors (Dal Bianco et al., 2011; Karapiperis and Verykios, 2015). This can be achieved by partitioning the set of all record pairs to be compared, for example, using blocking, and conducting the comparison of the different partitions in parallel on different processors.
5. **Advanced communication patterns** can be used to reduce the exponential growth of complexity. Such communication patterns include sequential, ring-by-ring, tree-based, and hierarchical patterns. Some of these patterns have been investigated for PPRL on multiple databases (Vatsalan et al., 2016).

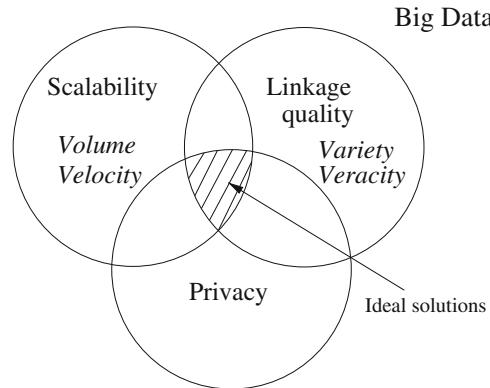
Examples of Application

Linking data is increasingly being required in a number of application areas (Christen, 2012). When databases are linked within a single organization, then generally privacy and confidentiality are not of great concern. However, linking data from several organizations imposes legal and ethical constraints on using personal data for the linkage, as described by the Australian Data-Matching Program Act (<https://www.legislation.gov.au/Details/C2016C00755>), the EU Personal Data Protection Act (<http://ec.europa.eu/justice/data-protection/>), and the HIPAA Act in the USA (<http://www.hhs.gov/ocr/privacy/>). PPRL is therefore required in several real applications, as the following examples illustrate:

1. **Health applications:** Several health applications ranging from health surveillance, epidemiological studies, and clinical trials to pub-

lic health research require PPRL as an efficient building block. For example, researchers are interested in aggregating health databases for quality health data mining. Health outbreak systems require data to be integrated from several sources, including human health, travel, and consumed drug, to allow the early detection of infectious diseases before they spread widely (Kelman et al., 2002). Similar patient matching is another application that requires linking different health databases, for clinical trials and customized patient care (Vatsalan and Christen, 2016).

2. **Government services and research:** The traditionally used small-scale survey studies have been replaced by linking databases to develop policies in a more efficient and effective way (Kelman et al., 2002). The research program “Beyond 2011” established by the Office for National Statistics in the UK, for example, aimed to study the options for production of population statistics for England and Wales, by linking anonymous data (Office for National Statistics, 2013). Social scientists use PPRL in the field of population informatics to study insights into our society (Kum et al., 2014).
3. **Business collaboration:** Many businesses take advantage of linking data across different organizations, such as suppliers, retailers, wholesalers, and advertisers, for improving efficiency, targeted marketing, and reducing costs of their supply chains. PPRL can be used for cross-organizational collaborative decision making which involves a great deal of private information that businesses are often reluctant to disclose (Zhu et al., 2017).
4. **National security applications:** PPRL techniques are being used by national security agencies and crime investigators to identify individuals who have committed fraud or crimes (Phua et al., 2012). These applications integrate data from various sources, such as law enforcement agencies, Internet service providers, the police, tax agencies, government services, as well as financial institutions to enable the accurate identification of crime and fraud or of terrorism suspects.



Privacy-Preserving Record Linkage, Fig. 3
Challenges of PPRL for Big Data

Future Directions for Research

In this section we describe the various open challenges and directions of PPRL for Big Data applications, as categorized in Fig. 3.

Scalability

The *volume* of data increases both due to the large size of databases and the number of different databases. Challenges of linking large databases have been addressed by using computational techniques to reduce the number of required comparisons between records, as described in section “Key Research Findings.” However, these techniques do not solve the scalability problem for Big Data completely, as they still result in exponential complexity. For example, even small blocks of records resulting from a blocking technique can still lead to a large number of comparisons between records with the increasing volume of data. Moreover, only limited work has been done to address the challenges of linking multiple databases.

Scalable blocking techniques are required that generate blocks of suitable sizes across multiple databases, as are advanced filtering techniques that effectively prune potential non-matches even further. With multiple databases, identifying subsets of records that match across only a subset of databases (e.g., patients in three out of five hospital databases) is even more challenging due to the large number of combinations

of subsets. Advanced communication patterns, distributed computing, and adaptive comparison techniques are required toward this direction for scalable PPRL applications. These techniques are largely orthogonal so that they can be combined to achieve maximal efficiency.

Another major aspect of Big Data is the *velocity*, i.e., the dynamic nature of data. Current techniques are only applicable in batch mode on static and well-defined relational data. Required are approaches for dynamic data, real-time integration, and data streams, for adaptive systems to link data as they arrive at an organization, ideally in (near) real time.

Linkage Quality

Big Data are often complex, noisy, and erroneous, which refer to the *veracity* and *variety* aspects. The linkage can be affected by the quality of data, especially in a privacy-preserving context where linkage is done on the masked or encoded version of the data. Masking data reduces the quality of data to account for privacy and therefore the quality of data would have an adverse effect on the whole PPRL process leading to low linkage quality. The trade-off between quality and privacy needs to be handled carefully for different privacy masking functions. The effect of poor data quality on the linkage quality becomes worse with increasing number of databases. Only two recent approaches support approximate matching for PPRL on multiple databases (Vatsalan and Christen, 2014; Vatsalan et al., 2016). More advanced classification techniques, such as collective (Bhattacharya and Getoor, 2007) and graph-based (Kalashnikov and Mehrotra, 2006) techniques, need to be investigated to address the linkage quality problem of PPRL.

Assessing the linkage quality in a PPRL project is very challenging because it is generally not possible to inspect linked records due to privacy concerns. Knowing the quality of linkage is crucial in many Big Data applications such as in the health or security domains. An initial work has been done on proposing interactive PPRL (Kum et al., 2014) where parts of sensitive

values are revealed for manual assessment in such a way that the privacy compromise is limited. Implementing such approaches in real applications is an open challenge that must be solved.

Privacy

Another open challenge in PPRL is how resistant the techniques are against different adversarial attacks. Most work in PPRL assumes the *semi-honest* adversary model (Lindell and Pinkas, 2009) and the trusted LU-based model. Furthermore, these works assume that the parties do not collude with each other (Vatsalan et al., 2013). Only few PPRL techniques consider the *malicious* adversary model as it imposes a high complexity (Vatsalan et al., 2013). More research is therefore required to develop novel security models that lie between these two models and prevent against collusion risks for PPRL.

Sophisticated attack methods (Christen et al., 2017; Kuzu et al., 2011) have been recently developed that exploit the information revealed during the PPRL protocols to iteratively gather information about sensitive values. Therefore, existing PPRL techniques need to be hardened to ensure they are not vulnerable to such attacks. Evaluation of PPRL techniques is challenged due to the absence of benchmarks, datasets, and frameworks. Different measurements have been used (Durham, 2012; Vatsalan et al., 2013, 2014), making the comparison of different PPRL techniques difficult. Synthetic datasets that are generated based on the characteristics of real data using data generators (Tran et al., 2013) have been used as an alternative to benchmark datasets. This limits the real evaluation of PPRL techniques to assess their application in the real setting.

Further, there has not been much interaction between practitioners and researchers of PPRL to allow better understanding of the whole data life cycle and to evaluate the applicability of PPRL in real applications. A comprehensive privacy strategy is essential including a closely aligned PPRL and privacy-preserving data technologies for strategic Big Data applications.

References

- Atallah M, Kerschbaum F, Du W (2003) Secure and private sequence comparisons. In: ACM WPES, pp 39–44
- Bhattacharya I, Getoor L (2007) Collective entity resolution in relational data. TKDD 1(1):5-es
- Boyd J, Randall S, Ferrante A (2015) Application of privacy-preserving techniques in operational record linkage centres. In: Gkoulalas-Divanis A, Loukides G (eds) Medical data privacy handbook. Springer, Cham
- Christen P (2012) Data matching. Data-centric systems and applications. Springer, Berlin/Heidelberg
- Christen P, Schnell R, Vatsalan D, Ranbaduge T (2017) Efficient cryptanalysis of bloom filters for PPRL. In: Kim J et al (eds) PAKDD. Springer, pp 628–640
- Dal Bianco G, Galante R, Heuser CA (2011) A fast approach for parallel deduplication on multicore processors. In: SAC. ACM, pp 1027–1032
- Durham EA (2012) A framework for accurate, efficient private record linkage. PhD thesis, Vanderbilt University, Nashville
- Office for National Statistics (2013) Matching anonymous data. In: Beyond 2011
- Kalashnikov D, Mehrotra S (2006) Domain-independent data cleaning via analysis of entity-relationship graph. TODS 31(2):716–767
- Karakasidis A, Koloniari G, Verykios VS (2015) Scalable blocking for PPRL. In: SIGKDD. ACM, pp 527–536
- Karapiperis D, Verykios V (2015) An LSH-based blocking approach with a homomorphic matching technique for PPRL. TKDE 27(4):909–921
- Karapiperis D, Vatsalan D, Verykios VS, Christen P (2015) Large-scale multi-party counting set intersection using a space efficient global synopsis. In: DAS-FAA
- Karapiperis D, Gkoulalas-Divanis A, Verykios VS (2017) Federal: a framework for distance-aware privacy-preserving record linkage. TKDE 30(2):292–304
- Kelman CW, Bass J, Holman D (2002) Research use of linked health data – a best practice protocol. ANZJPH 26:251–255
- Kum H, Krishnamurthy A, Machanavajjhala A, Ahalt S (2014) Social Genome: Putting Big Data to Work for Population Informatics. Computer 47(1):56–63
- Kum H-C, Krishnamurthy A, Machanavajjhala A, Reiter MK, Ahalt S (2014) Privacy preserving interactive record linkage. JAMIA 21(2):212–220
- Kuzu M, Kantarcioglu M, Durham E, Malin B (2011) A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In: PETTS, Waterloo. LNCS, vol 6794. Springer, pp 226–245
- Lindell Y, Pinkas B (2009) Secure multiparty computation for privacy-preserving data mining. JPC 1(1):59–98
- Phua C, Smith-Miles K, Lee V, Gayler R (2012) Resilient identity crime detection. IEEE TKDE 24(3):533–546
- Ranbaduge T, Vatsalan D, Christen P (2016) Scalable block scheduling for efficient multi-database record linkage. In: ICDM. IEEE, pp 1161–1166
- Randall SM, Ferrante AM, Boyd JH, Bauer JK, Semmens JB (2014) PPRL on large real world datasets. JBI 50:205–212
- Randall SM, Ferrante AM, Boyd JH, Brown AP, Semmens JB (2016) Limited privacy protection and poor sensitivity: is it time to move on from the statistical linkage key-581? HIMJ 45(2):71–79
- Schnell R, Bachteler T, Reiher J (2009) Privacy-preserving record linkage using Bloom filters. BMC MIDM 9(41):1–11
- Sehili Z, Kolb L, Borgs C, Schnell R, Rahm E (2015) PPRL with PPJoin. In: BTW, Hamburg
- Tran K-N, Vatsalan D, Christen P (2013) GeCo: an online personal data generator and corruptor. In: CIKM, San Francisco. ACM, pp 2473–2476
- Vatsalan D, Christen P (2012) An iterative two-party protocol for scalable PPRL. In: AusDM, CRPIT, Sydney, vol 134.
- Vatsalan D, Christen P (2014) Scalable PPRL for multiple databases. In: CIKM, Shanghai. ACM
- Vatsalan D, Christen P (2016) Privacy-preserving matching of similar patients. JBI 59:285–298
- Vatsalan D, Christen P, Verykios VS (2013) A taxonomy of PPRL techniques. JIS 38(6):946–969
- Vatsalan D, Christen P, O’Keefe CM, Verykios VS (2014) An evaluation framework for PPRL. JPC 6(1):35–75
- Vatsalan D, Christen P, Rahm E (2016) Scalable privacy-preserving linking of multiple databases using counting bloom filters. In: ICDMW PDDM, Barcelona. IEEE
- Zhu H, Liu H, Ou CX, Davison RM, Yang Z (2017) Privacy preserving mechanisms for optimizing cross-organizational collaborative decisions based on the Karmarkar algorithm. JIS 72:205–217

P

Private Data Integration

► Privacy-Preserving Record Linkage

Private Data Matching

► Privacy-Preserving Record Linkage

Private Record Linkage

► Privacy-Preserving Record Linkage

Probabilistic Data Integration

Maurice Van Keulen

Faculty of EEMCS, University of Twente,
Enschede, The Netherlands

Synonyms

Uncertain data integration

Definitions

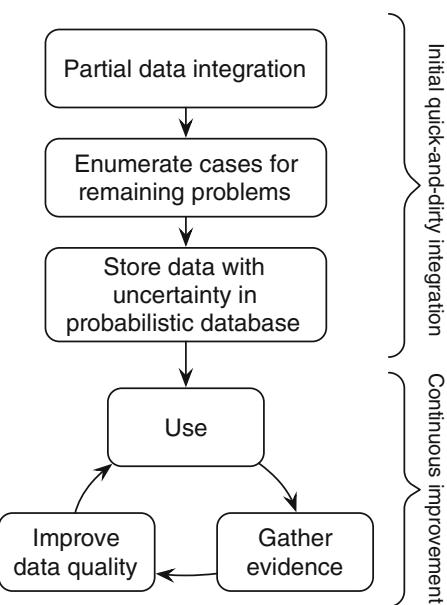
Probabilistic data integration (PDI) is a specific kind of data integration where integration problems such as inconsistency and uncertainty are handled by means of a probabilistic data representation. The approach is based on the view that data quality problems (as they occur in an integration process) can be modeled as uncertainty (van Keulen, 2012), and this uncertainty is considered an important result of the integration process (Magnani and Montesi, 2010).

The PDI process contains two phases (see Fig. 1): (i) a quick partial integration where certain data quality problems are not solved immediately, but explicitly represented as uncertainty in the resulting integrated data stored in a probabilistic database; (ii) continuous improvement by using the data – a probabilistic database can be queried directly resulting in possible or approximate answers (Dalvi et al., 2009) – and gathering evidence (e.g., user feedback) for improving the data quality.

A *probabilistic database* is a specific kind of DBMS that allows storage, querying, and manipulation of uncertain data. It keeps track of alternatives and the dependencies among them.

Overview

This chapter explains a special kind of data integration approach, called Probabilistic Data Integration. We first define the concept as well as the related notion of a Probabilistic Database. After presenting an illustrative example that is used



Probabilistic Data Integration, Fig. 1 Probabilistic data integration process (van Keulen and de Keijzer, 2009)

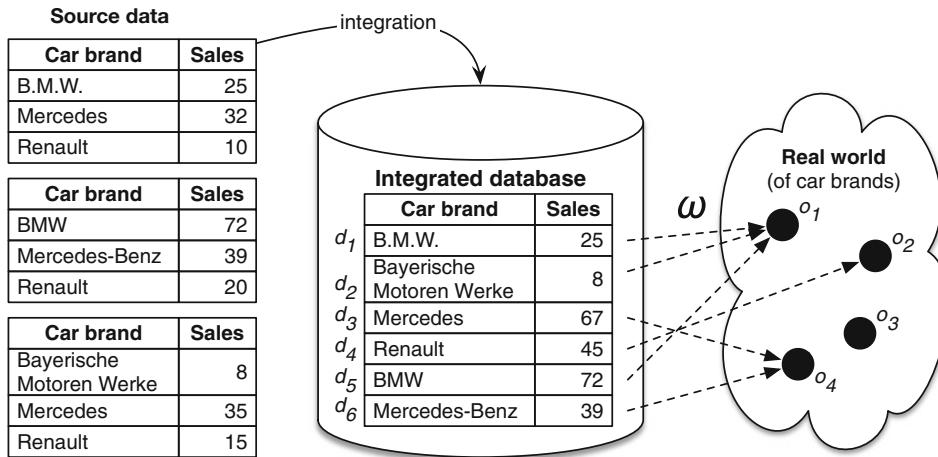
throughout the chapter, a motivation for the PDI approach is given based on its strengths in dealing with the main problems in data integration.

The technical part of this chapter first addresses the notion of a probabilistic database capable of storing and querying probabilistic representations, and then explains which probabilistic representations can be used for various data integration problems. We conclude with example applications and future developments.

Example

As a running example taken from van Keulen (2012), imagine a manufacturer of car parts supplying major car brands. For a preferred customer program (preferred customer defined as one with sales over 100) meant to avoid loosing important customers to competitors, data of three production sites needs to be integrated.

Figure 2 shows an example data integration result and a part of the real world it is supposed to represent. Observe that at first glance there is no preferred customer due to *semantic duplicates*: The “same” car brand occurs more than once



Probabilistic Data Integration, Fig. 2 An uncareful data integration leading to a database with semantic duplicates

under different names because of different conventions. Importantly, data items d_3 and d_6 refer to the same car brand, and their combined sales is 106, so “Mercedes-Benz” should be a preferred customer.

Typical data-cleaning solutions support duplicate removal that merges data items when they likely refer to the same real-world object, such as d_1 and d_5 merged into a new data item d_{15} ; $d_3, d_6 \mapsto d_{36}$ analogously. But, it is quite possible that an algorithm would not detect that also d_2 refers to “BMW.” Note that this seemingly small technical glitch has a profound business consequence: it determines whether “BMW” is considered a preferred customer or not, risking loosing it to a competitor.

What do we as humans do if we suspect that “BMW” stands for “Bayerische Motoren Werke”? *We are in doubt.* Consequently, humans simply consider both cases, reason that “BMW” *might* be a preferred customer and act on it if we decide that it is important and likely enough. It is this behavior of “doubting” and “probability and risk assessment” that probabilistic data integration is attempting to mimic.

Motivation

“Data integration involves *combining* data residing in different *sources* and providing users

with a *unified view* of them” (Lenzerini, 2002). Applications where uncertainty is unavoidable especially call for a probabilistic approach as the highlighted terms in the definition illustrate:

- It may be hard to extract information from certain kinds of sources (e.g., natural language, websites).
- Information in a source may be missing, of bad quality, or its meaning is unclear.
- It may be unclear which data items in the sources should be combined.
- Sources may be inconsistent complicating a unified view.

Typically, data integration is an iterative process where mistakes are discovered and repaired, analyses are repeated, and new mistakes are discovered... Still, we demand from data scientists that they act responsibly, i.e., they should know and tell us about the deficiencies in integrated data and analytical results.

Compared to traditional data integration, probabilistic data integration allows:

- postponement of solving data integration problems, hence, provides an initial integration result much earlier;
- better balancing of trade-off between development effort and resulting data quality;

- an iterative integration process with smaller steps (Wanders et al., 2015);
- leveraging human attention based on feedback; and
- more robustness being less sensitive to wrong settings of thresholds and wrong actions of rules (van Keulen and de Keijzer, 2009).

Probabilistic Databases

Probabilistic data integration hinges on the capability to readily store and query a voluminous probabilistic integration result as provided by a probabilistic database. The two main challenges of a probabilistic database is that it needs both to *scale* to large data volumes and also to do *probabilistic inference* (Dalvi et al., 2009).

The formal semantics is based on *possible worlds*. In its most general form, a probabilistic database is a probability space over the possible contents of the database. Assuming a single table, let I be a set of tuples (records) representing that table. A probabilistic database is a discrete probability space $PDB = (W, P)$, where $W = \{I_1, I_2, \dots, I_n\}$ is a set of possible instances, called possible worlds, and $P : W \rightarrow [0, 1]$ is such that $(\sum_{j=1..n} P(I_j)) = 1$.

In practice, one can never enumerate all possible worlds; instead a more concise representation is needed. Many representation formalisms have been proposed differing a.o. in expressiveness (see Panse 2015, Chp.3 for a thorough overview).

Figure 3 shows a probabilistic integration result of our running example of Fig. 2 where possible duplicates are probabilistically merged; see section ‘Record Level’. The used representation formalism is based on U-relations (Antova et al., 2008), which allows for dependencies between tuples, for example, tuples d_3 and d_6 (top left in Fig. 3) both exist together or are both absent.

Relational probabilistic database systems that, to a certain degree, have outgrown the laboratory bench include MayBMS (Koch, 2009; Antova et al., 2009), Trio (Widom, 2004), and MCDB (Jampani et al., 2008). MayBMS and Trio focus on tuple-level uncertainty where probabilities

are attached to tuples, while MCDB focuses on attribute-level uncertainty where a probabilistic value generator function captures the possible values for the attribute.

Besides probabilistic relational databases, probabilistic versions of other data models and associated query languages can be defined by attaching a probabilistic ‘sentence’ to data items and incorporating probabilistic inference in the semantics of the query language that adheres to the possible worlds semantics (Wanders and van Keulen, 2015). For example, several probabilistic XML (Abiteboul et al., 2009; van Keulen and de Keijzer, 2009) and probabilistic logic formalisms have been defined (Fuhr, 2000; Wanders et al., 2016; De Raedt and Kimmig, 2015).

Probabilistic Data Integration

In essence probabilistic data integration is about finding probabilistic representations for data integration problems. These are discussed on three levels: attribute value, record, and schema level.

Value Level

Inconsistency and ambiguity Integrated sources may not agree on the values of certain attributes, or it is otherwise unknown which values are correct. Some examples: text parsing may be ambiguous: in splitting my own full name ‘Maurice Van Keulen,’ is the ‘Van’ part of my first name or my last name? Differences in conventions: one source may use first name-last name (as customary in the West) and another last name-first name (as customary in China). Information extraction: is a phrase a named entity of a certain type or not?

In the formalism of the previous section, this is represented as:

	Firstname	Lastname	
d_1^a	Maurice	Keulen	$(r_4 \mapsto 0)$
d_1^b	Maurice	Van Keulen	$(r_4 \mapsto 1)$
d_2^a	Zhang	Li	$(r_5 \mapsto 0)$
d_2^b	Li	Zhang	$(r_5 \mapsto 1)$
d_3	Paris	Hilton	$(r_6 \mapsto 0)$

Probabilistic Data Integration, Fig. 3

Example of a probabilistic database (resulting from indeterministic deduplication of Fig. 2) with a typical query and its answer (Taken from van Keulen 2012)

PDB		Worlds	
car	sales	rva	P
d_1	25	$(r_1 \mapsto 0)$	$(r_1 \mapsto 0) 0.1$ ‘ d_1, d_2, d_5 different’
d_2	8	$(r_1 \mapsto 0)$	$(r_1 \mapsto 1) 0.6$ ‘ d_1, d_5 same’
d_5	72	$(r_1 \mapsto 0)$	$(r_1 \mapsto 2) 0.3$ ‘ d_1, d_2, d_5 same’
d_{15}	97	$(r_1 \mapsto 1)$	$(r_2 \mapsto 0) 0.2$ ‘ d_3, d_6 different’
d_2	8	$(r_1 \mapsto 1)$	$(r_2 \mapsto 1) 0.8$ ‘ d_3, d_6 same’
d_{125}	105	$(r_1 \mapsto 2)$	
d_4	45		$Q = \text{SELECT SUM(sales)}$
d_3	67	$(r_2 \mapsto 0)$	FROM carsales
d_6	39	$(r_2 \mapsto 0)$	WHERE sales ≥ 100
d_{36}	106	$(r_2 \mapsto 1)$	‘sales of preferred customers’

All possible worlds with their answer to Q

World descr.	World	Probability	Q
I_1 $(r_1 \mapsto 0), (r_2 \mapsto 0)$	$\{d_1, d_2, d_3, d_4, d_5, d_6\}$	$0.1 \cdot 0.2 = 0.02$	0
I_2 $(r_1 \mapsto 1), (r_2 \mapsto 0)$	$\{d_{15}, d_2, d_3, d_4, d_6\}$	$0.6 \cdot 0.2 = 0.12$	0
I_3 $(r_1 \mapsto 2), (r_2 \mapsto 0)$	$\{d_{125}, d_3, d_4, d_6\}$	$0.3 \cdot 0.2 = 0.06$	105
I_4 $(r_1 \mapsto 0), (r_2 \mapsto 1)$	$\{d_1, d_2, d_{36}, d_4, d_5\}$	$0.1 \cdot 0.8 = 0.08$	106
I_5 $(r_1 \mapsto 1), (r_2 \mapsto 1)$	$\{d_{15}, d_2, d_{36}, d_4\}$	$0.6 \cdot 0.8 = 0.48$	106
I_6 $(r_1 \mapsto 2), (r_2 \mapsto 1)$	$\{d_{125}, d_{36}, d_4\}$	$0.3 \cdot 0.8 = 0.24$	211

Possible answers

sum(sales)	P
0	0.14
105	0.06
106	0.56
211	0.24

Other derivable figures

description	sum(sales)	P
Minimum	0	0.14
Maximum	211	0.24
Answer most likely world	106	0.48
Most likely answer	106	0.56
Sec. most likely answer	211	0.24
Expected value	116.3	N.A.

where r_i ($i \in \{4, 5, 6\}$) govern the uncertainty which names are correct or preferred.

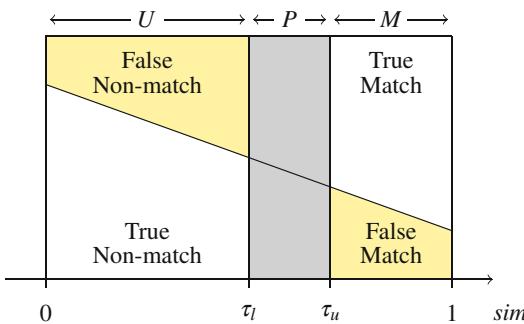
Data imputation A common approach to dealing with missing values is data imputation, i.e., using a most likely value and/or a value that retains certain statistical properties of the data set. Especially for categorical attributes, imputing with a wrong value can have grave consequences. In general, an imputation method is a classifier that predicts a most suitable value based on the other values in the record or data set. A classifier can easily not only predict one value but several possible ones each with an associated probability of suitability. By representing the uncertainty around the missing value probabilistically, the result is more informative and is more robust against imperfect imputations.

Record Level

Semantic duplicates, entity resolution

A semantic duplicate is almost never detected with absolute certainty unless both records are identical. Therefore, there is a gray area of record pairs that may or may not be semantic duplicates. Even if an *identifier* is present, in practice it may not be perfectly reliable. For example, it has once been reported in the UK that there were 81 million National Insurance numbers but only 60 million eligible citizens.

Traditional approaches for deduplication are based on pairwise tuple comparisons. Pairs are classified into *matching* (M) and *unmatching* (U) based on similarity, then clustered by transitivity, and, finally, merged by cluster. The latter may require solving inconsistencies (Naumann and Herschel, 2010).



Probabilistic Data Integration, Fig. 4 Grey area in tuple matching (Taken from Panse et al. 2013)

In such approaches with an absolute decision for tuples being duplicates or not, many realistic possibilities may be ignored leading to errors in the data. Instead, a probabilistic database can directly store an indeterministic deduplication result (Panse et al., 2013). In this way, all significantly likely duplicate mergings find their way into the database, and any query answer or other derived data will reflect the inherent uncertainty.

Indeterministic deduplication deviates as follows (Panse et al., 2013). Instead of M and U , a portion of tuple pairs are now classified into a third set P of *possible matches* based on two thresholds (see Fig. 4). For pairs in this *gray area*, both cases are considered: a match or not. Duplicate clustering now forms clusters for $M \cup U$ (in Fig. 2, there are 3 clusters: $\{d_1, d_2, d_5\}$, $\{d_4\}$, $\{d_3, d_6\}\}$. For each cluster, the possible worlds are determined, e.g., d_1, d_2, d_5 all different, d_1, d_5 the same and d_2 different, or d_1, d_2, d_5 all the same. To represent the probabilistic end result, a random variable is introduced for each cluster with as many values as possible worlds for that cluster, and merged and unmerged versions of the tuples are added according to the situation in the world. Figure 3 shows the end result.

A related problem is that of entity resolution (Naumann and Herschel, 2010). The goal of data integration is often to bring together data on the same real-world entities from different sources.

In the absence of a usable identifier, this matching and merging of records from different sources is a similar problem.

Repairs Another record-level integration problem is when a resulting database state does not satisfy some constraints. Here the notion of a *database repair* is useful. A repair of an inconsistent database I is a database J that is consistent and “as close as possible” to I (Wijsen, 2005). Closeness is typically measured in terms of the number of “insert,” “delete,” and “update” operations needed to change I into J . A repair, however, is in general not unique. Typically, one resorts to *consistent query answering*: the intersection of answers to a query posed on all possible repairs within a certain closeness bound. But, although there is no known work to refer to, it is perfectly conceivable that these possible repairs can be represented with a probabilistic database state.

Grouping data While integrating grouping data also inconsistencies may occur. A grouping can be defined as a membership of elements within groups. When different sources contain a grouping for the same set of elements, two elements may be in the same group in one source and in different groups in the other. Wanders et al. (2015) describe such a scenario with groups of orthologous proteins which are expected to have the same function(s). Biological databases like Homologene, PIRSF, and eggNOG store results of determining orthology by means of different methods. An automatic (probabilistic!) combination of these sources may provide a continuously evolving unified view of combined scientific insight of higher quality than any single method could provide.

Schema Level

Probabilistic data integration has been mostly applied to instance-level data, but it can also be applied on schema level. For example, if two sources hold data on entity types T and T' , and

these seem similar or related, then a number of hypotheses may be drawn up:

- T could have exactly the same meaning as T' ,
- T could be a subtype of T' or vice versa, or
- T and T' partially overlap and have a common supertype.

But it may be uncertain which one is true. It may even be the case that a hypothesis may only be partially true, for example, with source tables “student” and “PhD student.” In most cases, a PhD student is a special kind of student, but in some countries such as the Netherlands, a PhD student is actually an employee of the university. Also employees from a company may pursue a PhD. In short, not all tuples of table “PhD student” should be integrated into “student.” This also illustrates how this schema-level problem may be transformed into a record-level problem: a representation can be constructed where all tuples of a type probabilistically exist in a corresponding table. The uncertainty about two attributes being “the same” is an analogous problem.

Data Cleaning

Probabilistic data allows new kinds of cleaning approaches. High quality can be defined as a high probability for correct data and low probability for incorrect data. Therefore, cleaning approaches can be roughly categorized into uncertainty reducing and uncertainty increasing.

Uncertainty Reduction: Evidence If due to some evidence from analysis, reasoning, constraints, or feedback, it becomes apparent that some cases are definitely (not) true, then uncertainty may be removed from the database. For example, if in Fig. 3 feedback is given from which it can be derived that d_3 and d_6 are for certain the same car brand, then in essence $P(r_2 \mapsto 0)$ becomes 0 and $P(r_2 \mapsto 1)$ becomes 1. Consequently, all tuples that need

$(r_2 \mapsto 0)$ to be true to exist can be deleted (d_3 and d_6). Furthermore, random variable r_2 can be abolished, and the term $(r_2 \mapsto 1)$ can be removed from all probabilistic sentences. It effectively removes all possible worlds that contradict with the evidence. van Keulen and de Keijzer (2009) have shown that this form of cleaning may quickly and steadily improve quality of a probabilistic integration result.

If such evidence cannot be taken as absolutely reliable, hence cannot justify the cleaning actions above, the actual cleaning becomes a matter of massaging of the probabilities. For example, $P(r_2 \mapsto 1)$ may be increased only a little bit. In this approach, a probability threshold may be introduced above which the abovedescribed random variable removal is executed. As evidence accumulates, this approach converges to a certain correct database state as well, so data quality improvement is only slowed down provided that the evidence is for a large part correct.

Uncertainty Increase: Casting Doubt Perhaps counterintuitive, but increasing uncertainty may improve data quality, hence could be an approach for cleaning. For example, if due to some evidence it becomes unlikely that a certain tuple is correct, a random variable may be introduced, and possible repairs for tuples may be inserted. In effect, we are casting doubt on the data and insert what seems more likely. Consequently the uncertainty increases, but the overall quality may increase, because the probability mass associated with incorrect data decreases and the probability mass for correct data increases (assuming the evidence is largely correct).

Measuring uncertainty and quality The above illustrates that uncertainty and quality are orthogonal notions. Uncertainty is usually measured by means of entropy. Quality measures for probabilistic data are introduced by (van Keulen and de Keijzer, 2009): expected precision and expected recall. These notions are based on the intuition that the quality of a correct query answer is better if the system dares to claim that it is correct with a higher probability.

Example Applications

A notable application of probabilistic data integration is the *METIS* system, “an industrial prototype system for supporting real-time, actionable maritime situational awareness” (Huijbrechts et al., 2015). It aims to support operational work in domains characterized by constantly evolving situations with a diversity of entities, complex interactions, and uncertainty in the information gathered. It includes natural language processing of heterogeneous (un)structured data and probabilistic reasoning of uncertain information. METIS can be seen as an open-source intelligence (OSINT) application.

Another notable and concrete example of an existing system is *MCDB-R* (Arumugam et al., 2010). It allows risk assessment queries directly on the database. Risk assessment typically corresponds to computing interesting properties of the upper or lower tails of a query result distribution, for example, computing the probability of a large investment loss.

Probabilistic data integration is in particular suited for applications where much imperfection can be expected but where a quick-and-dirty integration and cleaning approach is likely to be sufficient. It has the potential of drastically lowering the time and effort needed for integration and cleaning, which can be considerable since “analysts report spending upwards of 80% of their time on problems in data cleaning” (Haas et al., 2015).

Other application areas include:

- Machine learning and data mining: since probabilistically integrated data has a higher information content than “data with errors,” it is expected that models of higher quality will be produced if probabilistic data is used as training data.
- Information extraction from natural language: since natural language is inherently ambiguous, it seems quite natural to represent the result of information extraction as probabilistic data.
- Web harvesting: websites are designed for use by humans. A probabilistic approach may lead

to more robust navigation. Subtasks like finding search results (Trieschnigg et al., 2012) or finding target fields (Jundt and van Keulen, 2013) are typically based on ranking “possible actions.” By executing not only one but a top- k of possible actions and representing resulting data probabilistically, consequences of imperfect ranking are reduced.

Future Developments

Probabilistic data integration depends on scalable probabilistic database technology. An important direction of future research is the development of probabilistic database systems and improving their scalability and functionality. Furthermore, future research is needed that compare the effectiveness of probabilistic data integration vs. non-probabilistic data integration approaches for real-world use cases.

Cross-References

- ▶ [Data Cleaning](#)
- ▶ [Data Deduplication](#)
- ▶ [Data Integration](#)
- ▶ [Graph Data Integration and Exchange](#)
- ▶ [Holistic Schema Matching](#)
- ▶ [Record Linkage](#)
- ▶ [Schema Mapping](#)
- ▶ [Semantic Interlinking](#)
- ▶ [Truth Discovery](#)
- ▶ [Uncertain Schema Matching](#)

References

- Abiteboul S, Kimelfeld B, Sagiv Y, Senellart P (2009) On the expressiveness of probabilistic xml models. VLDB J 18(5):1041–1064. <https://doi.org/10.1007/s00778-009-0146-1>
- Antova L, Jansen T, Koch C, Olteanu D (2008) Fast and simple relational processing of uncertain data. In: Proceedings of ICDE, pp 983–992
- Antova L, Koch C, Olteanu D (2009) $10^{(10^6)}$ worlds and beyond: efficient representation and processing

- of incomplete information. VLDB J 18(5):1021–1040. <https://doi.org/10.1007/s00778-009-0149-y>
- Arumugam S, Xu F, Jampani R, Jermaine C, Perez LL, Haas PJ (2010) MCDB-R: risk analysis in the database. Proc VLDB Endow 3(1–2):782–793. <https://doi.org/10.14778/1920841.1920941>
- Dalvi N, Ré C, Suciu D (2009) Probabilistic databases: diamonds in the dirt. Commun ACM 52(7):86–94. <https://doi.org/10.1145/1538788.1538810>
- De Raedt L, Kimmig A (2015) Probabilistic (logic) programming concepts. Mach Learn 100(1):5–47. <https://doi.org/10.1007/s10994-015-5494-z>
- Fuhr N (2000) Probabilistic datalog: implementing logical information retrieval for advanced applications. J Am Soc Inf Sci 51(2):95–110
- Haas D, Krishnan S, Wang J, Franklin M, Wu E (2015) Wisteria: nurturing scalable data cleaning infrastructure. Proc VLDB Endow 8(12):2004–2007. <https://doi.org/10.14778/2824032.2824122>
- Huijbrechts B, Velikova M, Michels S, Scheepens R (2015) Metis1: an integrated reference architecture for addressing uncertainty in decision-support systems. Proc Comput Sci 44(Supplement C):476–485. <https://doi.org/10.1016/j.procs.2015.03.007>
- Jampani R, Xu F, Wu M, Perez LL, Jermaine C, Haas PJ (2008) MCDB: a monte carlo approach to managing uncertain data. In: Proceeding of SIGMOD. ACM, pp 687–700
- Jundt O, van Keulen M (2013) Sample-based XPath ranking for web information extraction. In: Proceeding of EUSFLAT. Advances in intelligent systems research. Atlantis Press. <https://doi.org/10.2991/eusflat.2013.27>
- Koch C (2009) MayBMS: a system for managing large probabilistic databases. In: Aggarwal CC (ed) Managing and mining uncertain data. Advances in database systems, vol 35. Springer. https://doi.org/10.1007/978-0-387-09690-2_6
- Lenzerini M (2002) Data integration: a theoretical perspective. In: Proceeding of PODS. ACM, pp 233–246. <https://doi.org/10.1145/543613.543644>
- Magnani M, Montesi D (2010) A survey on uncertainty management in data integration. JDIQ 2(1):5:1–5:33. <https://doi.org/10.1145/1805286.1805291>
- Naumann F, Herschel M (2010) An introduction to duplicate detection. Synthesis lectures on data management. Morgan & Claypool. <https://doi.org/10.2200/S00262ED1V01Y201003DTM003>
- Panse F (2015) Duplicate detection in probabilistic relational databases. PhD thesis, University of Hamburg
- Panske F, van Keulen M, Ritter N (2013) Indeterministic handling of uncertain decisions in deduplication. JDIQ 4(2):9:1–9:25. <https://doi.org/10.1145/2435221.2435225>
- Trieschnigg R, Tjin-Kam-Jet K, Hiemstra D (2012) Ranking xpaths for extracting search result records. Technical report TR-CTIT-12-08, Centre for telematics and information technology (CTIT)
- van Keulen M (2012) Managing uncertainty: the road towards better data interoperability. IT – Inf Technol 54(3):138–146. <https://doi.org/10.1524/itit.2012.0674>
- van Keulen M, de Keijzer A (2009) Qualitative effects of knowledge rules and user feedback in probabilistic data integration. VLDB J 18(5):1191–1217
- Wanders B, van Keulen M (2015) Revisiting the formal foundation of probabilistic databases. In: Proceeding of IFSA-EUSFLAT. Atlantis Press, p 47. <https://doi.org/10.2991/fsa-eusflat-15.2015.43>
- Wanders B, van Keulen M, van der Vet P (2015) Uncertain groupings: probabilistic combination of grouping data. In: Proceeding of DEXA. LNCS, vol 9261. Springer, pp 236–250. https://doi.org/10.1007/978-3-319-22849-5_17
- Wanders B, van Keulen M, Flokstra J (2016) Judged: a probabilistic datalog with dependencies. In: Proceeding of DeLBP. AAAI Press
- Widom J (2004) Trio: a system for integrated management of data, accuracy, and lineage. Technical report 2004-40, Stanford InfoLab. <http://ilpubs.stanford.edu:8090/658/>
- Wijsen J (2005) Database repairing using updates. ACM TODS 30(3):722–768. <https://doi.org/10.1145/1093382.1093385>

Process Filtering

► Business Process Querying

Process Management

► Business Process Querying

Process Matching

► Business Process Model Matching

Process Mining

► Business Process Analytics

Process Model Repair

Abel Armas Cervantes

University of Melbourne, Melbourne, VIC,
Australia

Definitions

In the context of Process Mining, Process model repair is a model enhancement operation for reconciling a set of existing differences between an event log and a model. The aim of this operation is to reconcile the set of differences while preserving, as much as possible, the resemblance between the original and the repaired model.

Overview

Business processes are core assets in modern organizations. Oftentimes they are captured as models to ease their understanding for the involved participants and enable the identification and prevention of issues that could arise during their execution. In addition to their descriptive nature, process models can be also prescriptive since they can be used to automate the process executions. Thus, it is crucial that process models stay up to date and reflect the intended behavior.

Modern information systems supporting business processes can maintain detailed data about the executions of such processes. This data can be extracted as event logs, where every execution of a process is recorded as a trace, i.e., as a sequence of activity occurrences, a.k.a. *events*. While models describe or prescribe the expected behavior of a process, event logs capture the observed behavior. Thus, it can contain both positive deviations, e.g., work-arounds implemented by the participants of the process to improve process performance, and negative deviations, e.g., violations of compliance rules.

Conformance checking, one of the three main operations in process mining, compares the expected versus the observed behavior. A conformance checker takes a log and a model as input,

computes the best alignment between the behavior they represent, and, as output, spells out the differences between them. There exist exact and approximate conformance checking techniques, and they can be broadly categorized depending on the execution semantics they adopt (interleaving or true concurrency) or the type of feedback they generate (low-level or high-level feedback). The differences detected between a model and a log can be used to identify violations that should be corrected or work-arounds with a positive impact in the process performance that should be integrated into the model. A more extensive explanation about conformance checking can be found in the previous chapter.

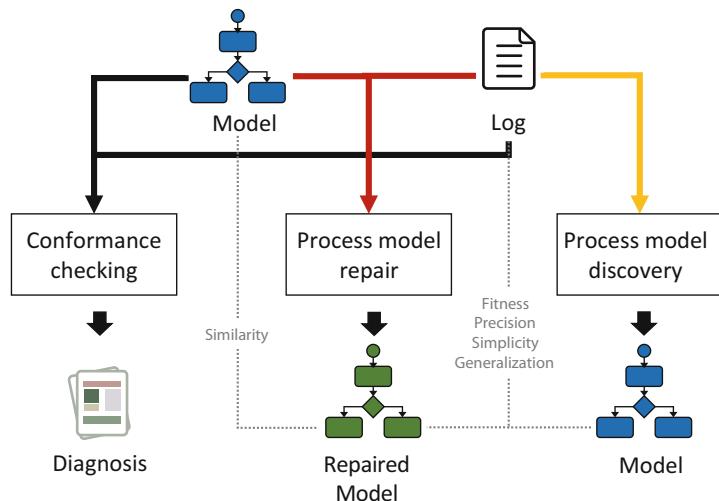
Process model repair, a type of *enhancement*, is another operation in process mining, which consists on modifying the model to better reflect reality, such that the performed modifications reconcile existing differences between the expected (model) and the observed behavior (log). Thus, process model repair can be seen as a step forward after the detection of differences via a conformance checker. Given the organic nature of organizations, as well as continuous changes that are external to the company, process model repair provides a way for maintaining models up to date and true to the processes they represent. Note, however, that the majority of the existing techniques have focused their attention on repairing control-flow problems, while little attention has been devoted to problems related to other aspects of the process, e.g., resources. This entry presents existing process model repair techniques and discusses existing challenges.

Process Model Repair in Context

Process model repair sits in-between two process mining operations: process model discovery and conformance checking, cf. Fig. 1. On the one hand, discovery techniques use the information captured in an event log for creating a process model from scratch. On the other hand, repair techniques take a (current) model as a baseline for the creation of a new version of such model that reconciles differences between the current

Process Model Repair,

Fig. 1 Process model repair, conformance checking, and process discovery



model and a log. In both operations, discovery and repair, there is a common point of reference for measuring the quality of the resulting model, its similarity with respect to the log. In order to assess the similarity between a model and a log, four main quality measures have been proposed: fitness, precision, simplicity, and generalization (see Buijs et al. 2012 and van der Aalst 2011 for more details about these measures).

The reasons to repair a model, instead of discovering a brand new one, can be manyfold: the model can provide additional insights into the process, is normative, or is used as a reference model. Thus, while the repair operation modifies the behavior of a baseline model, the repaired version has to resemble as much as possible the baseline model. Then, the structural similarity between the repaired and the baseline model has been taken into account as another way to assess the quality of a repair. The structural similarity between process models has been widely studied in the literature, for instance, Dijkman et al. (2011, 2009) present different strategies for the computation of the similarity between models based on graph edit distance.

Process model repair is a multi-perspective operation which considers the different aspects mentioned above: fitness, precision, generalization, simplicity, and similarity. As discussed later in the chapter, some techniques have devoted their attention to maximize exclusively fitness at

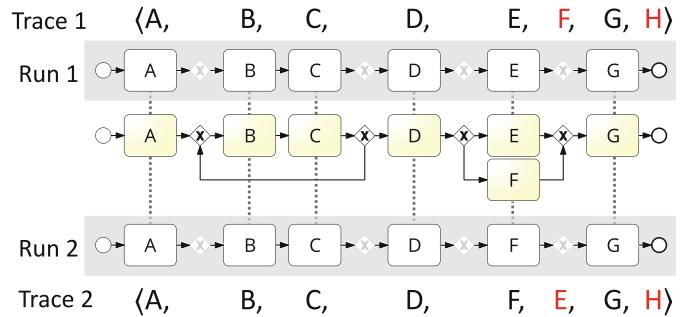
the cost of a negative impact in precision, while others have defined a set of repair patterns that aim at coping with the negative impact in precision of the repair at a cost of requiring human intervention.

Fitness-Based Repair

Fitness has been one of the main drivers for guiding the repair of a model. Intuitively, fitness measures the amount of behavior captured in a log that can be replayed by a given model; thus a model has perfect fitness iff it can replay every trace in the log. In the context of repair, given a model and a log, such that there exist some differences between them, a fitness-based repair modifies the model in such a way that it reconciles the existing differences and every trace can be replied in the resulting model.

The first process model repair technique given an event log was presented in Fahland and van der Aalst (2015). It is a fitness-based approach and uses three types of repair operations over the model: subprocess insertion, skip insertion, and cycle removal. This technique is divided into three steps; first an alignment between the observed and the expected behavior is computed. This alignment consists of finding for each trace in the log the most similar run of the process model, where the similarity is

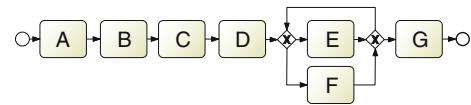
Process Model Repair,
Fig. 2 Alignment between
the behavior of a model
and a log



measured in terms of the events that can occur at the same execution state in both the trace and the run. This alignment is computed by a conformance checker. The second step finds the parts of the log that cannot be replayed by the model and gathers them into sub-logs. Finally, a model is discovered from each of the created sub-logs, which are then inserted into the original model. The repaired model has a perfect fitness and thus can fully replay the log.

Consider the example shown in Fig. 2 and a log with two traces: $\{\langle A, B, C, D, E, F, G, H \rangle, \langle A, B, C, D, F, E, G, H \rangle\}$, the central part of the picture depicts a process model in BPMN notation and its two possible *runs* (shaded areas), while the outer part displays the traces in the log *aligned* with the behavior of the model. The aligned traces show that in **Trace 1**, events *F* and *H* could not be matched with any activity occurrence in **Run 1**, whereas in **Trace 2**, events *E* and *H* could not be matched with any activity occurrence in **Run 2**. The repaired model produced by the technique in Fahland and van der Aalst (2015) is shown in Fig. 3. Observe that, in the repaired model, activities *B* and *C* cannot be repeated and activities *E* and *F* can occur any number of times one after the other.

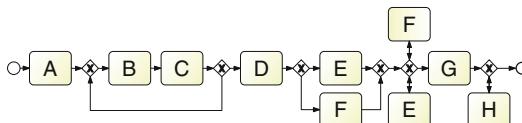
Dees et al. (2017) refine the technique presented in Fahland and van der Aalst (2015) by considering key performance indicators (KPI) when repairing the model. Prior the repair takes place, Dees et al. (2017) filter out the parts of the log that should be considered according to some specific KPI goals. Once the relevant parts of the log have been selected, the repair technique presented in Fahland and van der Aalst (2015) is



Process Model Repair, Fig. 3 Repaired process model by Fahland and van der Aalst (2015)

applied. The approach presented in Dees et al. (2017) is the first effort to integrate additional information to control-flow data while repairing a model.

Polyvyanyy et al. (2016) present a repair approach that detects the tasks that have to be inserted or deleted from the model in order to maximize fitness. The authors acknowledge the importance of other metrics in addition to fitness, though the identification of techniques to improve on such metrics during repair is left to future work. During the repair, this technique uses only two operations to modify the model: insertion of new tasks in a self-loop and creation of new paths for skipping a single task. In order to control the type of changes that can be performed over the model, this approach allows users to assign costs to the repair operations applied over specific events and, additionally, to specify the budget for the overall repair of the model. Thus, a model is repaired in such a way that it maximizes fitness, but the sum of the cost of all the operations over the events does not exceed the specified budget. The first step during the repair is to compute the alignment between the behavior of the model and the behavior of the log, and then finding the operations to be applied over the model. The repaired version of the running example (Fig. 2) by the technique in Polyvyanyy et al. (2016) is shown in Fig. 4.



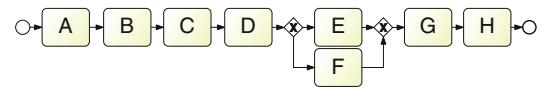
Process Model Repair, Fig. 4 Repaired process model by Polyvanyy et al. (2016)

Observe that the changes performed by the approaches above have a positive impact on fitness and on the similarity with the original model; however they affect other aspects. Specifically, the models in Figs. 3 and 4 contain additional behavior that is neither allowed in the original model nor observed in or implied by the log, which affects negatively the precision of the model. For example, in both repaired models, tasks *E* and *F* can be repeated any number of times, and similarly for task *H* in Fig. 4, however, the log does not contain any repetition of any task.

Generalized Repair

The approaches mentioned in the previous section have devoted their attention to the modification of the model while taking the fitness as the main driver. However, the choice of maximizing fitness in the repaired model comes with the assumption that all the observed behavior in the log is desired and shall be integrated into the model (positive deviances). The last is not always true since a log can be noisy, that is, it can contain undesired behavior or lack of relevant behavior that was not captured in the log (e.g., it can contain incomplete traces). In this context, Rogge-Solti et al. (2016) proposed a generalized conformance checking framework that combines three operations: conformance checking, process model repair, and inconsistency detection in the log.

The generalized conformance checking framework puts forward the idea of automatically repairing both the log and the model in one goal. The repaired log and model are as close as possible to those given as input, are fully fitting between them (i.e., the log has perfect fitness with the model), and are repaired according to



Process Model Repair, Fig. 5 Repaired process model by Rogge-Solti et al. (2016)

some levels of trust assigned to the model and the log given as input. The full trust in either the model or the log defines the ends of a spectrum. Specifically, if the log is fully trusted and not the model, then a process model discovery technique is applied, whereas, if the model is fully trusted and not the log, then a log is created by simulating the model.

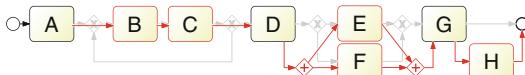
As an example, Fig. 5 displays the model – with the log $\{\langle A, B, C, D, E, G, H \rangle, \langle A, B, C, D, F, G, H \rangle\}$ – generated by the current implementation of the generalized conformance checking framework given the model and log in Fig. 2, and trust = ~ 0.5 for both, model and log.

Note that this technique does not seek to reconcile all the differences detected with respect to the log, as observed in the example above, but instead aims at integrating only the changes that lead to a model, similar to the original model, that respects the defined trust values.

P

Interactive and Incremental Repair

Different to the techniques presented so far, Armas Cervantes et al. (2017) introduce an alternative approach, which is interactive and incremental and that aims at supporting the users during the reconciliation of differences. The foundation of this approach is the visualization of the differences detected by a conformance checker, which offers a suggestion on how to reconcile an existing difference. The conformance checker used in this approach is the one presented in García-Bañuelos et al. (2015); it adopts true concurrency semantics – thus it is able to find differences involving concurrency – and explains the differences using predefined behavioral patterns. The differences detected by this conformance checker are then explained to the user by contrasting the behavior occurring in the log but not in the model



Process Model Repair, Fig. 6 Interactive and incremental repair: visualization of the differences

and vice versa. For instance, given the model and the log in Fig. 2, the conformance checker can diagnose that *In the model, the tasks E and F are mutually exclusive, while in the log they occur in parallel.*

The interactive and incremental repair translates every difference detected by the conformance checker into a graphical representation that suggest a way for reconciling such difference. Then, these visualizations are given to the users as recommendations on how to perform a repair. This approach gives the users full control on what and how to repair and does not make any assumption on how relevant certain differences can be for the users; instead, it ranks the detected differences according to the impact in the observed behavior. Specifically, a difference with high impact is the one involving highly frequent behavior in the log. Note that it is up to the user to decide what is the aspect they want to improve (fitness, precision, generalization, etc.) during the repair. Clearly, the user in charge of repairing of the model needs to have a deep understanding about the process, since the integration of negative deviances would be undesired.

Figure 6 shows the visualizations provided to the user for the running example depicted in Fig. 2. In this graphical representation, the elements in red have to be inserted or are those affected by the difference, whereas the elements in gray have to be deleted.

Future Directions

The presented techniques range from fully automatic approaches, where every difference between the model and the log is fixed, to interactive and incremental approaches, where the users ultimately choose what and how to apply a repair. A future direction is to explore options

for balancing automatic and manual operations, e.g., by considering compliance rules attached to the model. For instance, a difference could be automatically reconciled in the model iff the repair does not violate any of the given compliance rules; otherwise, the differences can be presented to the users, so that they can decide how to tackle the issue.

Another future direction, for both automatic and nonautomatic approaches, is to consider a balance of the various quality aspects (fitness, precision, generalization, simplicity, and similarity to the original model) during the repair. Finally, existing techniques can be extended by considering other data – such as KPIs as in Dees et al. (2017) or resources – in addition to control-flow information for repairing a model according to predefined business goals.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Conformance Checking](#)

References

- Armas Cervantes A, van Beest NRTP, Rosa ML, Dumas M, García-Bañuelos L (2017) Interactive and incremental business process model repair. Springer International Publishing, pp 53–74
- Buijs JCAM, van Dongen BF, van der Aalst WMP (2012) On the role of fitness, precision, generalization and simplicity in process discovery. Springer, Berlin/Heidelberg, pp 305–322. https://doi.org/10.1007/978-3-642-33606-5_19
- Dees M, de Leoni M, Mannhardt F (2017) Enhancing process models to improve business performance: a methodology and case studies. Springer, pp 232–251. https://doi.org/10.1007/978-3-319-69462-7_15
- Dijkman R, Dumas M, García-Bañuelos L (2009) Graph matching algorithms for business process model similarity search. Springer, Heidelberg/Berlin, pp 48–63. https://doi.org/10.1007/978-3-642-03848-8_5
- Dijkman R, Dumas M, van Dongen B, Käärlik R, Mendling J (2011) Similarity of business process models: metrics and evaluation. Inf Syst 36(2):498–516. <https://doi.org/10.1016/j.is.2010.09.006>
- Fahland D, van der Aalst WM (2015) Model repair – aligning process models to reality. Inf Syst 47: 220–243. <https://doi.org/10.1016/j.is.2013.12.007>

- García-Bañuelos L, van Beest NR, Dumas M, La Rosa M (2015) Complete and interpretable conformance checking of business processes. Technical report, BPM Center. <http://eprints.qut.edu.au/91552/>
- Polyvyanyy A, Aalst WMPVD, Hofstede AHMT, Wynn MT (2016) Impact-driven process model repair. ACM Trans Softw Eng Methodol 25(4):1–60. <https://doi.org/10.1145/2980764>
- Rogge-Solti A, Senderovich A, Weidlich M, Jan-Mendling, Gal A (2016) In log and model we trust? a generalized conformance checking framework, Springer International Publishing, pp 179–196
- van der Aalst W (2011) Process mining. Discovery, conformance and enhancement of business processes. Springer-Verlag Berlin Heidelberg

Process Retrieval

- ▶ Business Process Querying

Process Variant Discovery

- ▶ Trace Clustering

Process Warehousing

- ▶ Multidimensional Process Analytics

Programmable Memory/Processing in Memory (PIM)

- ▶ Active Storage

Provisioning and Decommissioning

- ▶ Elasticity

Python

Sandeep Nagar

GD Goenka University, Gurgaon, Haryana, India

Definitions

Python programming language is an open-source, portable, high-level general-purpose programming language. It features an interpreter which provides interactive environment, dynamic-type system, as well as automatic memory management. Being object oriented in nature, it is widely used and provides a large and comprehensive library for real-world applications. Python 2 and Python 3 (<https://www.python.org/downloads/>) are two versions of Python interpreters being presently used. Python 3 is not back-compatible with Python 2, but it is gaining ground among developers and will ultimately replace Python 2 entirely. Python gained popularity among data scientist due to availability of easy-to-use libraries and ease of working with variety of file format in both local and remote locations.

High-Level Programming

Python is a general-purpose high-level programming language. It provides easy access to library (called *modules* here (<https://docs.python.org/3/py-modindex.html>)) functions for performing the low-level tasks. Python code is converted to byte code which in turn is interpreted byte by byte by the Python interpreter (<https://docs.python.org/3/tutorial/interpreter.html>). This makes the system portable as Python interpreter converts Python code (with few or no modifications) as per target processor on a target machine. The interpreter itself is quite configurable, which gives power to developer to customize the same as per project requirements.

Interactive Environment

Python interpreter provides two modes of operations: *shell mode* and *program mode*. In *shell*

mode, a REPL (Read-Evaluate-Print-Loops) is provided which:

- Reads a Python expression typed by a user.
- Evaluates Python expression.
- Prints out the return value after evaluation.
- Loops back and does it all over again.

REPL-based workflow is useful for interactive development in data science since it shortens the overall development time by providing facility for easier debugging. In addition to allowing quick and easy evaluation of Python statements, it also showcases a searchable history (press up and down keys) and tab completion, defining key bindings for ease of use and easy access to help documentation.

In *program mode*, one can define a Python program via a text editor of user's choice and run by simply invoking the Python interpreter and feeding the program file. A variety of IDE (Integrated Development Environments) provides rich feature of syntax highlighting and file management in this regard. IPython and IPython notebook environments also provide rich feature to teach Python effectively where code, textual and graphical contents, can be integrated within a single environment (a browser in the case of a notebook). Since many programming language kernels are now supported here, the name has been changed to Jupyter Notebook. This facility also provides easier way for teamwork where Jupyter Notebooks can be deployed on server to work collectively in an interactive manner.

Object-Oriented Programming

Most of small Python programs are procedural in nature, i.e., a set of procedures were defined to compute a problem, and flow of information was controlled within these procedures to obtain a desired output. On the other hand, larger Python programs are framed under an object-oriented programming (OOP) framework which deals with data as an object on which different

methods act, to produce a desired result. All computable entities are treated as an object whose nature is defined via functions called methods. A class of objects can be clubbed and defined together to create an environment similar to a physical one to define an easy to understand correspondence to real-life situations. This is critical for data science because different data can be treated as different objects as per requirements which give flexibility as well as intuitive understanding of physical features represented by the data.

Multipurpose

OOP-based architecture and open-source nature of Python enabled developers from a variety of domains and enriched Python ecosystem with their fields of expertise. One can define a module (a set of classes) specific for different domains of studies. Some of them are mentioned in Table 1.

Big data workflow usually employs modules related to networking, database management, machine learning, statistics, plotting, as well as making GUI for final products. The Pandas module (<https://pandas.pydata.org/>) is used extensively by Big Data community in general as an alternative to R. Python provides a good option for such a workflow which can be stitched together with ease.

Extensible

Instead of providing all functionalities in core Python program, it was designed to be extensible, i.e., user can choose to have functionality as per requirements. For example, if somebody needs to work on statistical problems, then one can simply use that particular module, ignoring others!

Python programs can be easily embedded in programs written in programming languages like Julia, C, C++, R, etc. On the other hand, other programming language codes can also be embedded into Python. For example, the modules cython and jython are used to embed C and Java codes. This has enabled the use of a lot of legacy code written in other languages and already optimized for speed and stability. This

Python, Table 1 List of areas and corresponding Python modules

Field of study	Name of Python module
Scientific computation	scipy, numpy, sympy
Networking	networkx
Cryptography	pyOpenSSL
Game development	PyGame
Graphic user interface (GUI)	pyQT
Mobile app	kivy
Machine learning	scikit-learn, tensorflow
Statistics	pandas
Image processing	scikit-image
Plotting	Matplotlib
Database	SQLAlchemy
HTML and XML parsing	BeautifulSoup
Natural language processing	nltk
Testing	nose

avoids replication of efforts and increases productivity of an organization tremendously. This is critical for a data scientist since legacy codes in tools like R and C/C++ have already been optimized and hence need not be rewritten. Python can compete both as a stand-alone option and the one where other packages can be embedded in the overall workflow (Louridas and Ebert, 2013).

Minimalistic

Minimalistic design philosophy emphasizes code *readability* as a priority for language design. It provides a syntax structure that allows programmers to express concepts in fewer lines of code compared with other high-level languages. Python features a dynamic-type system where Python interpreter guesses the *type* of an object by its definition, which avoids a user's need to define the same explicitly. Indentations are used to define a block of statements for loops and functions instead of enclosing them in braces like in C/C++. Writing loops and functions is intuitive in nature. These results accelerated pace of development and ease of understanding. Some example below can highlight these features. If one wishes to print a string "Hello World," then one simply writes a Python expression as:

```
>>>print ("Hello World")
```

Even looping over a set of numbers can be written quite intuitively. For example, generating square roots of first ten numbers can simply be defined as:

```
1 >>>import numpy as np
2 >>>num = np.arange(1:11)
3 >>>for i in num:
4     answer=np.sqrt(num)
5     print (answer)
```

Data science needs such a design to define complex problems in an easier manner. Also different parts of a problem can be designed with different team members which can be stitched together within a single Python code. Minimalistic design feature enabled fewer lines of codes and hence easier understanding.

P

Data Types

Python provides a rich ecosystem of data types as shown in Table 2. Note that some data structures are defined in core Python program, whereas others are defined in modules, and hence they can be accessed by adding module name succeeded by a dot operator. Big data applications need ability to define classes of objects with flexible and easier definitions and operations. Table 2 lists some such objects which are used frequently.

Python, Table 2 List of areas and corresponding Python modules

Data type	Features
Integers	<code>int()</code>
Floating point numbers	<code>float()</code>
Complex numbers	<code>complex(a,b)</code> or <code>a+jb</code>
Immutable list = Tuple	<code>tuple()</code>
Heterogeneous list	<code>list()</code>
Array of numbers	<code>numpy.ndarray()</code>
Sets	<code>set()</code>
Key-value pairs	<code>dict()</code>
1D tabular data	<code>pandas.Series()</code>
Multidimensional tabular data	<code>pandas.DataFrame()</code>
Tensors	<code>tensorflow.Tensor()</code>

User-defined data structures can be defined using `Class` object definition and defining associated methods which imply associated properties. The ability to do so with ease is one of the major forces behind widespread utility of Python for big data applications.

Summary

Python is among the world's most preferred language among developers due to its open-source, flexible, portable, and user-friendly architecture. It is used by school students (Elkner, 2000; Grandell et al., 2006) as well as computer scientists for both simple and complex calculations (<https://ganga.web.cern.ch/ganga/>) with equal ease of effort to define a problem (Blank et al., 2004). With ever-increasing rich library of modules, it will continue to dominate the computing world in the near future too. Big data analytics community has embraced Python (Cielen et al., 2016; McKinney, 2012) for ease of

use and extensible nature as well as other features defined above.

References

- Blank D, Kumar D, Meeden L, Yanco H (2004) Pyro: a Python-based versatile programming environment for teaching robotics. *J Educ Res Comput (JERIC)* 4(3):3
- Cielen D, Meysman A, Ali M (2016) Introducing data science: big data, machine learning, and more, using Python tools. Manning Publications Co., Shelter Island
- Elkner J (2000) Using Python in a high school computer science program. In: Proceedings of the 8th international Python conference, pp 2000–2001
- Grandell L, Peltomäki M, Back RJ, Salakoski T (2006) Why complicate things? Introducing programming in high school using Python. In: Proceedings of the 8th Australasian conference on computing education, vol 52, ACE'06. Australian Computer Society, Inc., Darlinghurst, pp 71–80. <http://dl.acm.org/citation.cfm?id=1151869.1151880>
- Louridas P, Ebert C (2013) Embedded analytics and statistics for big data. *IEEE Softw* 30(6):33–39. <https://doi.org/10.1109/MS.2013.125>
- McKinney W (2012) Python for data analysis: data wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc., Sebastopol

Q

Quality Assessment

- ▶ Data Quality and Data Cleansing of Semantic Data

Query Optimization Challenges for SQL-on-Hadoop

Mohamed A. Soliman
Datometry Inc., San Francisco, CA, USA

Definitions

In database management systems, the query optimizer is the component responsible for mapping an input query to the most efficient mechanism of executing the query, called query execution plan. Query execution is a resource-intensive operation that consumes memory, I/O, and network bandwidth resources of the underlying database management system. Query optimizer builds a space of plan alternatives capturing the different ways of executing an input query, such as different orderings of joins among the tables referenced by the query. Each plan alternative is assessed using a cost model that computes a cost estimate reflecting a prediction of the plan's wall clock running time. The optimizer picks the most efficient execution plan according to such cost estimates.

Overview

The job of a query optimizer is to turn a user query into an efficient query execution plan. The optimizer typically generates the execution plan by considering a large space of possible alternative plans and assigning an estimated cost to each alternative in order to pick the cheapest one. The art of query optimization is a combination of technically challenging problems (e.g., plan enumeration, cost estimation, statistics derivation, property enforcement, and join ordering); some of them are known to be NP-hard.

Query optimizer is one of the most performance-sensitive components in a database system. Differences in query plans may result in several orders of magnitude of difference in query performance, significantly more than any other contributing factor. Big data has triggered a renewed interest in query optimization techniques as a set of principled methods that could, when combined with scalable hardware resources and robust execution engines, make analyzing and processing petabytes of data finally possible. The increased amounts of data that need to be processed in today's world stress on the importance of building an intelligent query optimizer at the core of any query engine.

The impact of having a good optimizer on query performance is known to be substantial. The extensive research done by the database community in query optimization over the past decades provides a plethora of techniques that

can be leveraged and adopted in multiple big data processing environments such as Hadoop and the Cloud.

This article presents a number of query optimization research problems and technical challenges. The architectures of example big data query optimizers are described. The main similarities and differences of discussed optimizers are highlighted, and the technical problems tackled by each optimizer are illustrated.

Technical Challenges

Building a query optimizer is not an easy undertaking. It takes years of hard work to put together an industry-grade query optimizer that can be relied on in production settings. The amount of work devoted to the design and implementation of a new query optimizer is usually a considerable share of the effort needed to build a data management platform. This section gives an overview of some of the technical challenges involved in the design of query optimizers in big data query engines.

Modular Design

Building query optimizer as a complex monolithic software component does not allow adapting the optimizer design to the changing processing environments of big data. New data formats, storage structures, and query execution techniques are being constantly adopted in big data domains. The optimizer design needs to be modular to be easily pluggable in such evolving data management platforms. A crucial interface to query optimization engine is metadata acquisition. The optimizer makes extensive use of metadata (e.g., table/index definitions, statistical summaries such as histograms, and synopses) during plan enumeration, transformation rules, cardinality derivation, and cost computation. Using a highly extensible abstraction of metadata and system capabilities is important to allow query optimizer to deal with new data types and processing engine characteristics. Designing the optimizer's cardinality and cost models as pluggable components is important to allow

extending the optimizer with new advanced statistical models and cost estimation techniques.

Extensibility

Having all elements of a query and its optimizations as first-class citizens impacts optimizer architectures to a great extent. When an optimizer is designed based on a strict set of optimizations and query elements, adding new optimizations becomes technically hard. This often leads to resorting to multiple optimization phases, where the decisions made in earlier phases are revisited in later phases.

One example of new requirements that were triggered by big data processing environments is data distribution. In big data environments, scalability is realized by distributing data across many machines in a computing cluster. Optimizers ported from older database systems, or the ones that adopted older designs, typically handle data distribution by adding a plan parallelization phase, where data distribution is considered after the fact when some optimization decisions have been already made.

Multiphase optimizer design impacts the quality of final query plan since the available information does not impact all optimizer's decisions. Multiphase optimizers are notoriously difficult to extend as new optimizations or query constructs often do not match the predefined phase boundaries.

Addressing this challenge by abstracting query elements, data properties, and query optimizations as first-class constructs that optimizer treats similarly can avoid the problems of multiphase optimization where certain constructs are dealt with as an afterthought.

Exploiting Multi-core Architectures

Query optimization is probably the most CPU-intensive operation that a query engine performs. The space of query plan alternatives is combinatorial by definition. There are pressing needs for efficient exploration and pruning of the plan space in order to produce high-quality execution plans. Exploiting multi-core architectures allows optimizers to scale. By dividing the optimization

tasks into a set of work units, using multiple CPU cores for query optimization becomes possible.

A crucial challenge in this respect is formulating and capturing dependencies among optimizer's work units and designing a work unit scheduler that assigns optimization tasks to processing cores and efficiently gathers the computation results of these units to generate the optimization plan space.

Testing Optimizer Accuracy

Evaluating the accuracy of query optimizers objectively is a difficult problem. Benchmarks developed for assessing the query performance test the system as a whole end to end. However, no benchmarks are currently available to test a query optimizer in isolation. Being able to compare the accuracy of optimizers across different products independently is highly desirable.

The most performance-critical element in a cost-based optimizer is probably the accuracy of its cost model as it determines how to prune inferior plan alternatives. Typically, the cost units used in the cost model and displayed with the final plan do not reflect the anticipated wall clock time but are used only for comparison of alternative plans pertaining to the same input query. Comparing this cost value with the actual execution time does not give definitive conclusions about the accuracy of the cost model.

Optimization results are highly system-specific and therefore defy the standard testing approach where results are compared to a reference or baseline to check if the optimizer finds the “correct” solution. The optimal query plan for System A may widely differ from that for System B because of implementation differences in the query executors and the optimizers. These differences can lead to choosing radically different plans. Building a testing infrastructure for evaluating query optimizers is an important research problem.

Examples of Query Optimizers in Big Data Environments

This section presents example optimizers that are effectively used in production settings in different

big data management platforms. The main design principles behind these optimizers are given. The technical challenges in the previous section are also discussed in the context of each optimizer.

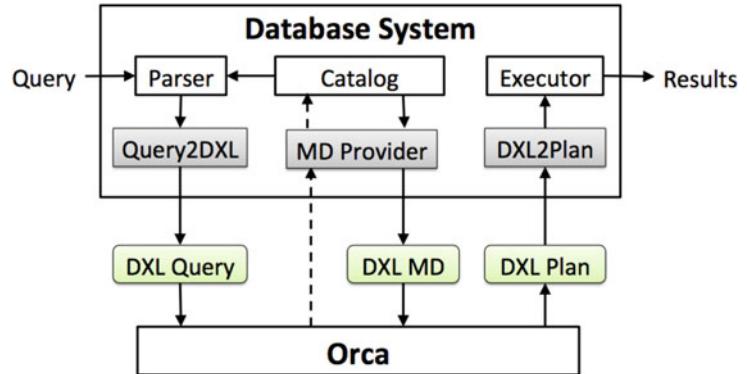
Orca

Orca (Orca Open Source 2018; Antova et al. 2014; El-Helw et al. 2015; Soliman et al. 2014) is the query optimizer of Pivotal’s data management products, including the massively parallel processing database system Greenplum (Pivotal 2018a) and the SQL-on-Hadoop database system HAWQ (Pivotal 2018b). While many query optimizers are tightly coupled with their host systems, a unique feature of Orca is its ability to run outside the database system as a stand-alone optimizer. This ability is crucial for supporting query engines with different computing architectures (e.g., massively parallel processing (MPP) and Hadoop) using one optimizer. It also allows leveraging the extensive legacy of relational optimization in new query processing paradigms. Furthermore, running the optimizer as a stand-alone product enables elaborate testing without going through the monolithic structure of a database system.

Figure 1 shows the interaction between Orca and an external database system. The database system needs to include translators that consume/emit data using Data eXchange Language (DXL), which is an XML-based language that is used to define an interface for accessing Orca. The Query2DXL translator converts a query parse tree into a DXL query, while the DXL2Plan translator converts a DXL plan into an executable plan. The implementation of such translators is done completely outside Orca, which allows multiple systems to interact with Orca by providing the appropriate translators.

The input to Orca is a DXL query. The output of Orca is a DXL plan. During optimization, the database system can be queried for metadata (e.g., table definitions). Orca abstracts metadata access details by allowing the database system to register a metadata provider (MD provider) that is responsible for serializing metadata into DXL before being sent to Orca. Metadata can also be

Query Optimization Challenges for SQL-on-Hadoop, Fig. 1
 Interaction between Orca and database system (Soliman et al. 2014)



consumed from regular files containing metadata objects serialized in DXL format.

The design of Orca is based on the Cascades optimization framework (Graefe 1995). The space of plan alternatives generated by the optimizer is encoded in a compact in-memory data structure called the Memo. The Memo structure consists of a set of containers called groups, where each group contains logically equivalent expressions. Memo groups capture the different sub-goals of a query (e.g., a filter on a table or a join of two tables). Group members, called group expressions, achieve the group goal in different logical ways (e.g., different join orders). Each group expression is an operator that has other groups as its children. This recursive structure of the Memo allows compact encoding of a huge space of possible plans.

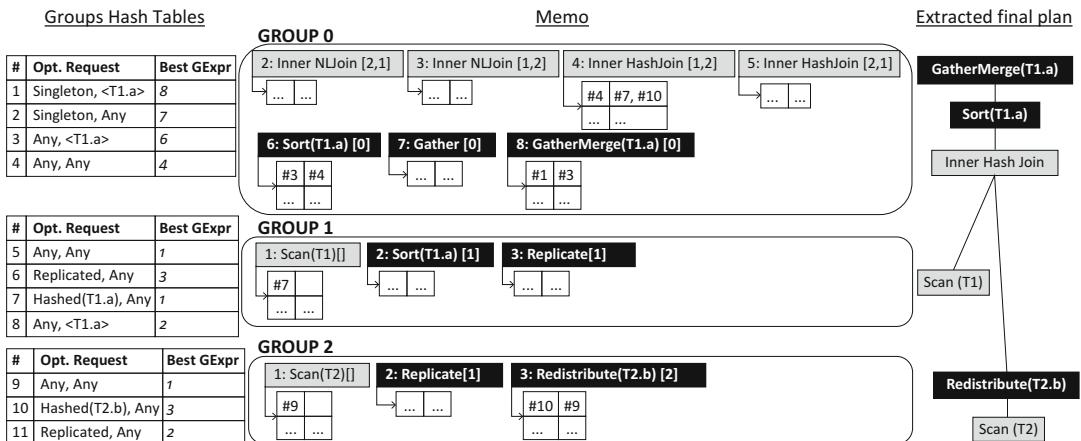
Orca optimizes queries in top-down fashion by computing optimization requests in the Memo. Each optimization request specifies physical properties (e.g., data distribution and sort order) that need to be satisfied. A Memo group is optimized under a given optimization request by finding the best plan, rooted at the group, that satisfies the required properties at the least cost.

Figure 2 shows how Orca processes optimization requests in the Memo for a simple join query between two tables T1 and T2 based on the join condition ($T1.a = T2.b$). In this example, the query results are required to be sorted on column T1.a. Assume that relation T1 is hash-distributed on column T1.a and relation T2 is hash-distributed on column T2.a. The initial op-

timization request is req. #1: (Singleton, T1.a), which specifies that query results are required to be gathered to a single node (typically the master node) based on the order given by column T1.a. The group hash tables are shown, where each request is associated with the best group expression that satisfies it at the least estimated cost. The black boxes indicate enforcer operators that are plugged in the Memo to deliver sort order and data distribution:

- Gather operator gathers results from all nodes to the master node.
- GatherMerge operator gathers sorted data from all nodes to the master node while keeping the sort order.
- Redistribute operator distributes tuples across nodes based on the hash value of given argument.
- Sort operator enforces a given sort order to the partition of tuples residing on each node.

Figure 3 shows the detailed optimization of req. #1 by InnerHashJoin[1,2], which is a hash-based algorithm for joining two relational inputs given by groups 1 and 2. For the join condition ($T1.a = T2.b$), one of the alternative plans is aligning child distributions based on join condition so that tuples to be joined are colocated. This is achieved by requesting Hashed(T1.a) distribution from group 1 and Hashed(T2.b) distribution from group 2. Both groups are requested to deliver any sort order.



Query Optimization Challenges for SQL-on-Hadoop, Fig. 2 Processing optimization requests in Orca (Soliman et al. 2014)

To satisfy a hashed distribution requirement, a hash value is computed for each tuple and used it to choose a node where the tuple is sent to. This allows parallel query execution on different nodes/machines. After child best plans are found, InnerHashJoin combines child properties to determine the delivered distribution and sort order. Note that the best plan for group 2 needs to hash distribute T2 on T2.b, since T2 is originally hash distributed on T2.a, while the best plan for group 1 is a simple scan, since T1 is already hash-distributed on T1.a.

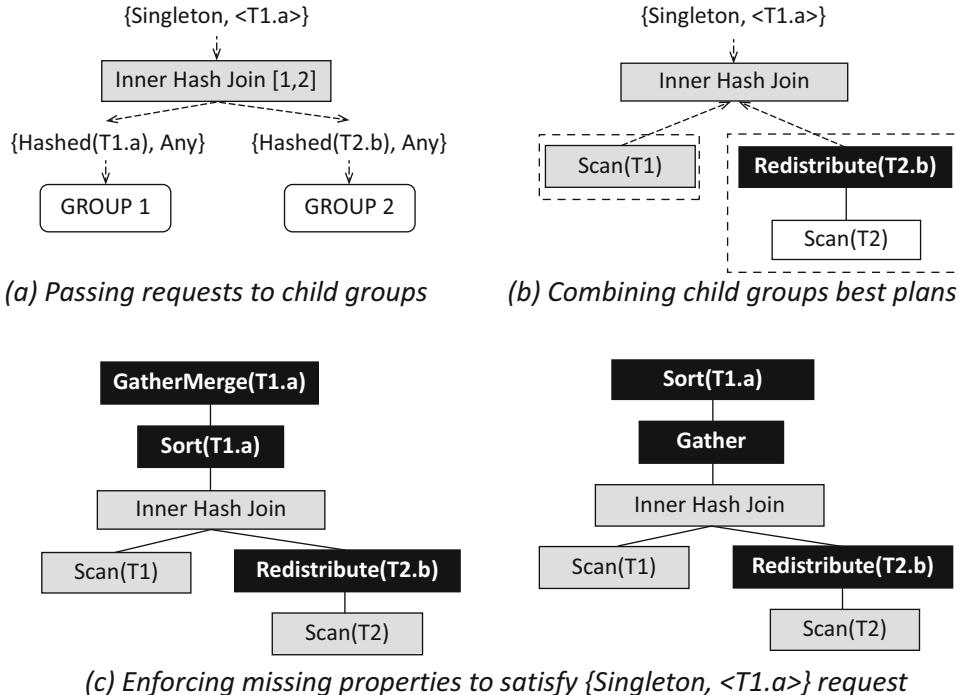
When it is determined that delivered properties do not satisfy the initial requirements, unsatisfied properties have to be enforced. Property enforcement in Orca is a flexible framework that allows each operator to define the behavior of enforcing required properties based on the properties delivered by child plans and operator local behavior.

Enforcers are added to the group containing the group expression being optimized. Figure 3c shows two possible plans that satisfy req. #1 through property enforcement. The left plan sorts join results on different nodes and then gather-merges sorted results at the master node. The right plan gathers join results from different nodes to the master node and then sorts them. These different alternatives are encoded in the Memo, and it is up to the cost model to differentiate their costs.

Finally, the best plan is extracted from the Memo based on the linkage structure given by optimization requests. Figure 2 illustrates plan extraction in Orca. The local hash tables of relevant group expressions are shown. Each local hash table maps incoming optimization request to corresponding child optimization requests. The best group expression of req. #1 is first looked up in the root group, which leads to GatherMerge operator. The corresponding child request in the local hash table of GatherMerge is req #3. The best group expression for req #3 is Sort. Therefore, GatherMerge is linked to Sort. The corresponding child request in the local hash table of Sort is req. #4. The best group expression for req. #4 is InnerHashJoin[1,2]. Sort is thus linked to InnerHashJoin. This procedure is followed to complete plan extraction leading to the final plan shown in Fig. 2.

Catalyst

Catalyst (Armbrust et al. 2015) is the query optimizer of SparkSQL, which is used to turn SQL queries into Spark execution plans. Many constructs in Catalyst are represented using tree data structures, including logical expressions, intermediate expression, and physical query execution plans. The optimizer design is based on decoupling the optimization phases which gives



Query Optimization Challenges for SQL-on-Hadoop, Fig. 3 Plan generation in Orca (Soliman et al. 2014)

room for including further optimizations as the design evolves.

The design of Catalyst is based on a query rewrite engine that runs batches of transformation rules. Each transformation rule converts an input tree to an equivalent output tree. Transformation rules are partitioned into batches. Batches are run sequentially, and rules within each batch are also run sequentially. The rule engine terminates when reaching a fixed point, which means that applying further transformations would not change the input expression anymore, or when a maximum number of iterations is reached. Query optimization in Catalyst takes place in a number of sequential phases, as depicted in Fig. 4.

The different phases that Catalyst optimizer goes through to produce the final query execution plan are next discussed. Figure 5 shows an example query which returns the *name* attribute of the Hive table *people* based on an equality condition applied to the *location* attribute.

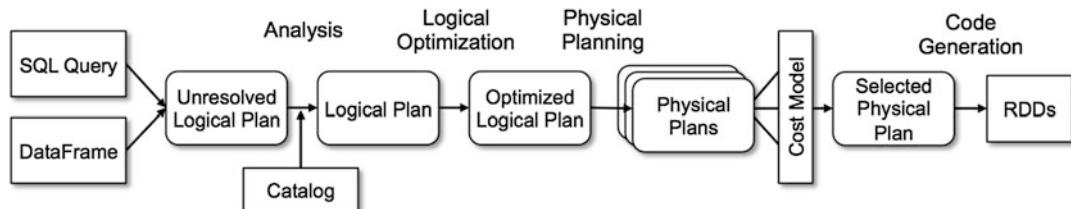
Query parsing and metadata lookup. Since the example query references a Hive table, a

HiveQL-based parser is used to generate an abstract syntax tree for the incoming query. Figure 5 shows the tree representation. The abstract syntax tree is preprocessed by resolving table names referenced in the query by looking up the MetaStore to obtain table definitions. A MetaStoreRelation is created for the unresolved relation in the query after looking up the MetaStore.

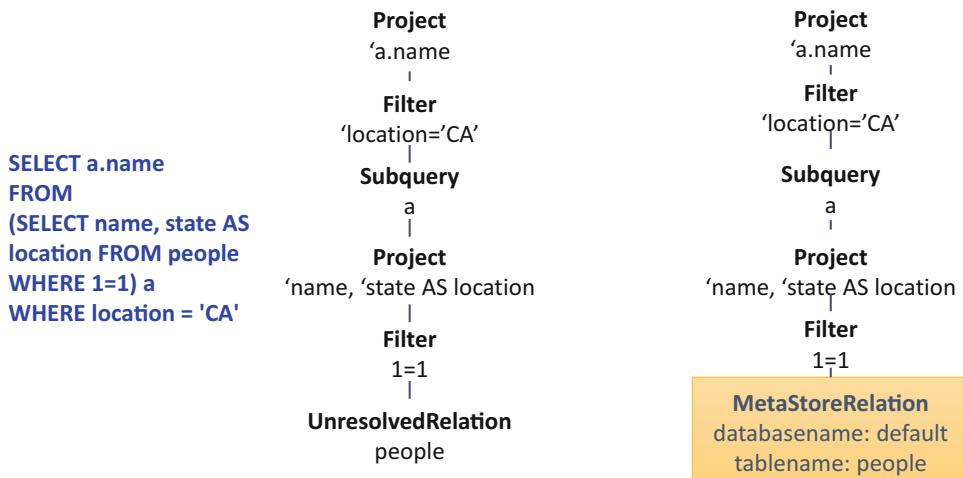
Resolving references. A globally unique identifier is assigned to each attribute (column) referenced in the query tree. Then, derivation of data types of every data item in the query tree takes place. Figure 6 shows unique numerical identifiers assigned to columns in the query tree.

Logical planning. Multiple logical rewrites are conducted to simplify the input query tree. These rewrite operations include the following:

- Removing redundant query operators.
- Constant folding, where inlined constant expressions are reduced to simple atomic values.



Query Optimization Challenges for SQL-on-Hadoop, Fig. 4 Query optimization phases in SparkSQL (Armbrust et al. 2015)



Query Optimization Challenges for SQL-on-Hadoop, Fig. 5 Query example in SparkSQL (Armbrust et al. 2015)

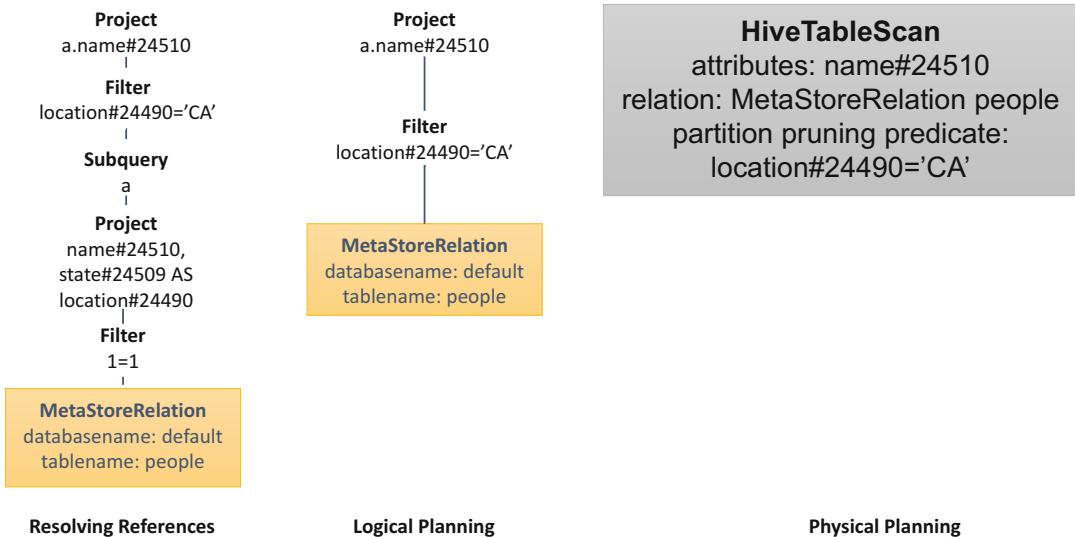
- Filter simplification, where trivial filters are removed and subtrees that are known to return no results are pruned.
- Filter push down so that filtering operations are applied as early as possible.
- Eliminate unused column references that are not needed to compute the final query answers.

Figure 6 shows the transformed query tree after applying the previous logical rewrites. For example, the Filter operator has been pushed down, while the Subquery operator was eliminated because of being a redundant operator.

Physical planning. In this phase, the optimized logical plan is transformed into a number of possible physical plans. This is done by recognizing operators in the logical plan that can be directly transformed into physical counterparts.

For example, a filter on a MetaStore relation can be transformed into HiveTableScan. In addition, a list of strategy objects, each of which can return a list of physical plan options, is used to transform logical operators to equivalent physical ones. Examples of strategy objects include HashJoin and NestedLoopJoin strategies. If a given strategy object is unable to plan all of the remaining operators in the tree, it creates a placeholder to be filled by other strategy objects. Figure 6 shows the final physical plan, which is reduced to a simple HiveTableScan with an embedded filter.

Plan parallelization. In this phase, data exchange operators are plugged into the generated physical plans to establish the required data distributions. Distribution requirements could arise from operator local requirements or from query-specific requirements. Physical operators may not be able to establish data distributions on their



Query Optimization Challenges for SQL-on-Hadoop, Fig. 6 Query example in SparkSQL (Armbrust et al. 2015)

own. In this case distribution enforcers are used to guarantee delivering the required data distribution.

Impala

The query optimizer of Impala (Kornacker and Erickson 2012) generates execution plans by following a two-phase approach:

- Phase 1 (single-node planning): In this phase, a query plan that runs on a single node (i.e., without considering data partitioning in the Hadoop cluster) is generated.
- Phase 2 (plan parallelization): In this phase, a parallelized query plan that processes distributed data in the cluster is generated.

In the first optimization phase, the query parse tree is translated into a non-executable single-node plan tree. The single node plan consists of different types of operators including table scan, join, aggregation, sort, and analytic evaluation functions. In this phase, filters specified in the given query are pushed down to be applied as close as possible to data sources. This is important to early-prune parts of the data that are not needed to compute the final query answer. New

filters could also be inferred from existing filters to prune data more aggressively.

Another important optimization that takes place in this phase is join ordering, where an efficient evaluation order of relational joins is identified. Heuristics are typically used to avoid exhaustive enumeration of the join orderings space. In the second optimization phase, the single-node plan is turned into a distributed (parallel) execution plan. The optimization objective of this phase is to minimize data movement and maximize scan locality.

Data movement is controlled by adding Exchange operators between plan nodes and by adding extra non-exchange plan nodes to minimize data movement across the network (e.g., local aggregation nodes). During this second phase, the join strategy for every join node is decided, including broadcast (one input of the join is replicated on all nodes) and partitioned/hashed (the two join inputs are partitioned across nodes using the join expression so that tuples that would join together reside on the same node).

All aggregations are executed as a local aggregation followed by a merge aggregation operation. The local aggregation output is partitioned on the grouping expressions, and the merge aggregation is done in parallel on all nodes. Sort is

parallelized similarly using a local sort followed by a single-node merge operation.

At the end of the second phase, the distributed plan tree is split into a set of plan fragments at the boundaries given by the exchange operators. Each plan fragment constitutes a unit of execution encapsulating a portion of the plan tree that operates on the same data partition on a single machine.

Apache Calcite

The Calcite project (Apache Calcite 2018) provides a Java-based SQL parser, optimizer, and data management framework that allows users to parse, optimize, and execute queries on a variety of data sources consumed using third-party connectors. A query is represented using a tree of operators-based relational algebra constructs. The operator tree is generated automatically by parsing SQL query. It can also be built manually using a set of given Java APIs.

Transformation rules are used to modify a relational algebra expression into an equivalent expression with the same semantics. This is needed to extend the search space of the optimizer while looking for the best execution plan of a given query. Transformations are guided by a cost model that performs the transformation if it yields a plan alternative with small cost according to the underlying cost model. The optimization framework is extensible allowing users to add new operators, transformation rules, and cost models.

Calcite provides two planner engine implementations: a volcano-style planner implementation that uses dynamic programming to exhaustively search the plan space and rule-based planner that builds the plan space by applying a fixed set of transformation rules.

Future Directions for Research

Despite a plethora of available commercial and open-source query optimizers, the effort needed to equip a data management platform with a scalable and sophisticated query optimizer remains considerable. Optimizer design remains to a large

extent tightly coupled with data management system components. Building a generic query optimization as service is still an open problem. Some efforts have been done toward this direction by abstracting optimizers interface and providing optimizer builder SDKs.

Big data applications and use cases are spreading into different environments and platforms including on-premises massively parallel processing systems and Hadoop ecosystems as well as public and private Cloud platforms. In such hybrid and rapidly changing environments, building query optimization techniques that are able to optimize queries across heterogeneous computing platforms is an important future direction of research.

Providing virtualized access to big data sources is gaining more traction with the wide adoption of the Cloud and the need to query data sources hosted by disparate environments. Performing query optimization within data virtualization layer poses new challenges regarding abstracting the characteristics and capabilities of the underlying data sources and seamless cross-optimization of queries against different query processing engines without disrupting existing application workflows.

Q

Cross-References

- ▶ [Impala](#)
- ▶ [Parallel Processing with Big Data](#)
- ▶ [Spark SQL](#)

References

- Antova L, El-Helw A, Soliman MA, Gu Z, Petropoulos M, Waas F (2014) Optimizing queries over partitioned tables in MPP systems. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data
- Apache Calcite (2018) <https://calcite.apache.org>

- El-Helw A, Raghavan V, Soliman MA, Caragea G, Gu Z, Petropoulos M (2015) Optimization of common table expressions in MPP database systems. Proc VLDB Endow 8:1704–1715
- Graefe G (1995) The cascades framework for query optimization. IEEE Data Eng Bull 18(3):19–29
- Kornacker M, Erickson J (2012) Cloudera impala: real-time queries in Apache Hadoop, for real. <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html>
- Orca Open Source (2018) <https://github.com/greenplum-db/gporca>
- Pivotal (2018a) Greenplum database. <http://greenplum.org/>
- Pivotal (2018b) HAWQ. <http://hawq.incubator.apache.org/>
- Soliman MA, Antova L, Raghavan V, El-Helw A, Gu Z, Shen E, Caragea GC, Garcia-Alvarado C, Rahman F, Petropoulos M, Waas F, Narayanan S, Krikellas K, Baldwin R (2014) Orca: a modular query optimizer architecture for big data. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data

Query Processing – k NN

Jianzhong Qi and Rui Zhang
The University of Melbourne, Melbourne, VIC, Australia

Synonyms

K Nearest Neighbor Query; k NN Query

Definitions

Consider a set of n data objects $O = \{o_1, o_2, \dots, o_n\}$ and a query object (user) q . Each object including the query object is associated with a d -dimensional vector representing its coordinate in a d -dimensional space ($d \in \mathbb{N}_+$). Given a query parameter k ($k \in \mathbb{N}_+$), the k nearest neighbor (k NN) query computes a size- k subset $S \subseteq O$ that contains the data objects that are the nearest to q :

$$\forall o_i \in S, o_j \in O \setminus S : \text{dist}(q, o_i) \leq \text{dist}(q, o_j)$$

Here, $\text{dist}(\cdot)$ is a function that returns the distance between two objects. A variety of distance functions have been considered in the literature, such as *Euclidean (L_2) distance*, *Manhattan (L_1) distance*, *Chebyshev (L_∞) distance*, spatial network distance, and *general metric distance*.

k NN query processing in the spatial data management context has focused on two- or three-dimensional Euclidean space and spatial network space.

Overview

The rapid growth of *location-based services* (LBS) has accumulated a massive amount of spatial data such as mapping data and user trajectories. Popular LBS such as Google Maps are serving millions of users who issue queries such as *finding Alice her nearest petrol stations*, which is a typical example of k NN queries. Most k NN algorithms use *spatial indices* over the data objects to help prune the unpromising search space and obtain high query efficiency. When the amount of data becomes too large to be indexed or queried on a single machine, parallelism needs to be exploited to overcome such limitations. This raises new challenges of k NN queries in parallel index management and query processing.

Key Research Findings

We start with a brief review of two classic k NN algorithms and then focus on the discussion of recent developments on k NN algorithms that use parallelism to manage the growing amount of data.

Traditional k NN Algorithms

The two arguably most popular k NN query algorithms are the *best-first* (Hjaltason and Samet 1995) and the *depth-first* (Roussopoulos et al. 1995) algorithms. These two algorithms rely on an tree index (e.g., R-trees Guttman 1984) over the data set O . They traverse the tree index starting from the tree root to find data objects nearest to q . During the traversal, the distance to the

k th NN among the data objects already visited is used to facilitate the pruning of unpromising tree branches that do not index nearer data objects. When a tree node N is visited, the best-first algorithm inserts all child nodes of N into a queue Q prioritized by an estimated distance between each child node and q . The next node to be visited is the one in Q with the smallest estimated distance to q . The depth-first algorithm simply visits the first child of each node before visiting the sibling nodes. While these two algorithms have shown high empirical query efficiency due to the pruning power of the tree indices, they are difficult to be parallelized because of the backtracking procedure in traversal.

Parallel k NN Algorithms

Parallel k NN query algorithms that suit popular parallel computation models such as *MapReduce* (Dean and Ghemawat 2008) have been developed to handle data beyond the processing capability of individual machines. Such algorithms partition the data space with a grid (a.k.a. *tiling*, cf. Fig. 1) so that data objects in different grid cells can be examined in parallel. A two-stage procedure is used in query processing. In the first stage, the cell C_q that contains the query object q is identified, and k NN is computed over the data objects in C_q (e.g., the 2NNs in Fig. 1 are o_3 and o_4). In the second stage, the cells adjacent to C_q are examined to find data objects that may be nearer to q than the k th

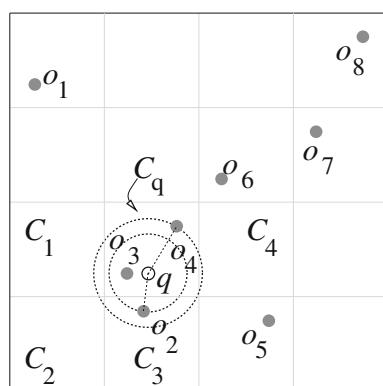
NN already found (e.g., cells C_1 , C_2 , C_3 , and C_4 , which are within the circle defined by the second NN o_4 ; o_2 is found to be the new second NN). This step may be repeated over more cells adjacent to the cells examined, until the nearest data object in these adjacent cells is farther away than the k th NN already found.

Hadoop-Based k NN Algorithms

Hadoop is an open-source implementation of the MapReduce computation model. The MapReduce model processes a task (a MapReduce job) in two phases: *Map* and *Reduce*. The Map phase assigns data to different machines in a cluster, while the Reduce phase runs computation on each machine. The data storage component of Hadoop named *Hadoop Distributed File System* (HDFS) is an implementation of the *Google File System* (Ghemawat et al. 2003). HDFS is optimized for large data blocks (e.g., 64 MB per block). Hadoop-based k NN algorithms need to design indices following this block size.

Aji et al. (2012) propose a k NN algorithm based on MapReduce with an assumption that the number of data objects is relatively small and the entire data set can be replicated to each machine for query processing. They focus on efficient processing of a large number of k NN queries at the same time, which is done by a single MapReduce job. In the Map phase, the query objects are partitioned with a grid and assigned to different machines according to their grid cell id. In the Reduce phase, the data set is replicated to each machine, upon which a local index (e.g., an R-tree) is built. The queries are then processed on each machine independently.

SpatialHadoop (Eldawy and Mokbel 2013) is a MapReduce framework for spatial query processing. It employs a two-level index structure that consists of a *global index* and a *local index*. The global index is built by partitioning the data space such that every partition can fit in a block in HDFS. Grid-based and R-tree-based indices are implemented in SpatialHadoop. A grid-based index simply partitions the data space with a uniform grid. An R-tree-based index creates an R-tree on a sample of the data set, where the leaf nodes are used to partition the data space.



Query Processing – k NN, Fig. 1 Grid-based k NN computation

Minimum bounding rectangles (MBR) of the partitions are stored in the global index which is kept in the main memory of the master node of the cluster. A local index is built on every partition and kept in HDFS. k NN queries are processed with the two-stage search space expansion procedure as described earlier. First, the partition containing the query object is identified using the global index, where a k NN query is computed using the local index. Then, adjacent partitions are explored until no more NNs can be found.

AQWA (Aly et al. 2015) builds a global index using the k-d tree (Bentley 1975) to create the data partitions where every partition is stored in HDFS as a block. The k-d tree is stored in the main memory of the master node. Additionally, the master node keeps a count of data objects for each partition. This enables processing a k NN query with only one MapReduce job as follows. The partitions nearest to the query object q that together contain at least k objects can be identified from the k-d tree index and the data counts. The farthest point of these partitions from q defines a circle centered at q . Only partitions within this circle needs to be fetched for query processing, which can be done by a single MapReduce job.

HBase-Based k NN Algorithms

HBase is an open-source *key-value* store (*NoSQL* database) built on top of HDFS. It hides the underlying block storage from users, and hence k NN algorithms based on it do not have to consider how to organize the data in blocks.

MD-HBase (Nishimura et al. 2011) is a multidimensional data store built on HBase. This system stores the *Z-order* (Orenstein and Merrett 1984) value of a data object together with the data object itself as a key-value pair in HBase. The key-value pairs are partitioned by the key range, and every partition is stored as a *region*. Data in HBase is indexed with a B^+ -tree-like index structure, where the key range of a region is stored in an index layer called the *META* layer. MD-HBase uses the longest common prefix of the keys (in the binary form) in a region as the index key of

the region. k NN queries are processed following a search space expansion procedure. At start, the region with an index key that shares the longest common prefix with the Z-order value of the query object q is identified, and an initial k NN answer is computed from the region. The k th NN found defines a distance from q where the search should expand to. This procedure repeats until no new k NN objects can be found in the expanded search space.

The *KR⁺ index* (Hsu et al. 2012) follows a similar k NN query procedure to that of MD-HBase but uses an R^+ -tree (Sellis et al. 1987) to group the data objects and Hilbert-order (Lawder and King 2001) values of the MBRs of the tree leaf nodes as index keys.

HGrid (Han and Stroulia 2013) uses another index key formulation, which is based on a two-level grid structure. The first level uses a coarse grid where the grid cells are indexed by the Z-order values. The second level further partitions each first-level grid cell with a regular grid where the cells are indexed by their row and cell numbers. The key value of a data object is formed by a concatenation of the index keys of the first and second level cells that this object locates at. HGrid has a slightly different k NN algorithm. The algorithm starts by estimating a region that may contain sufficient data objects to form a k NN answer based on the data density. Then, an initial k NN answer is computed from this region, and the search space expansion strategy is used to refine the answer.

COWI and *CONI* (Cahsai et al. 2017) build a quad-tree (Finkel and Bentley 1974) over the data set where the leaf node numbers are used as the index keys of the data objects. COWI stores the non-leaf levels of the quad-tree index in the memory of a master machine. The index also contains the number of data objects in each leaf node, which is used to help query processing in a way similar to that of AQWA. CONI considers the case where the quad-tree is too large to be held in memory. It stores the tree as a separate table in HBase. At query time, the tree index is accessed first to determine the leaf nodes that may be relevant for query processing.

Spark-Based k NN Algorithms

Spark provides an in-memory computation cluster framework to handle massive data. A set of data to be processed is modeled as a *resilient distributed dataset* (RDD). RDDs are stored in partitions across machines in the cluster.

GeoSpark (Yu et al. 2015) encapsulates RDDs with an abstraction named the *spatial resilient distributed dataset* (SRDD) to store spatial data. An SRDD is partitioned using a uniform grid. Local indices such as quad-trees may be created for data objects in the partitions. k NN query processing on GeoSpark is done by following the grid-based search space expansion strategy as discussed earlier.

Simba (Xie et al. 2016) uses both global and local indices. It creates a new RDD abstraction called the *IndexRDD* to store a local index (e.g., an R-tree) together with the data objects in each partition, while the partitions are created by the *STR* algorithm (Leutenegger et al. 1997). The global index (e.g., an R-tree) is created over the partitions and held in-memory in the master node for fast query processing. k NN queries on Simba also follow the search space expansion strategy but vary from the standard procedure slightly by fetching multiple partitions at start instead of only the partition containing the query object. The k th NN from the fetched partitions is used to define the search space to be explored next.

Examples of Application

k NN queries have a large variety of applications in different fields such as spatiotemporal databases, location-based services, and data mining, just to name but a few. In spatial data management, a typical application is in digital mapping, e.g., Google Maps, where k NN queries may be used for finding the nearest *points-of-interest* (POI) for query users. A closely related application is in ride sharing, e.g., Uber, where k NN queries may be used to find the nearest cars for car riders. Another application is in augmented reality gaming, e.g., Pokémon GO, where k NN queries may be used to find the nearest gaming objects of game players.

Future Directions for Research

The techniques discussed above consider stationary data objects whose locations do not change over time. With the growing popularity of smart mobile devices, location-based services for moving data objects (users) become prevalent, which call for efficient k NN algorithms over massive sets of moving objects (Nutanong et al. 2008; Wang et al. 2014; Li et al. 2014, 2016; Gu et al. 2016) and streaming location data (Koudas et al. 2004). Simply recomputing the query whenever there is an object location update may be infeasible due to the larger number of objects and updates. Keeping the query result updated under such constraints is a challenging task.

k NN queries in higher-dimensional spaces (Jagadish et al. 2005) is another challenge. As a major application domain, data mining requires k NN computation over feature spaces that may have thousands of dimensions. The techniques discussed are inapplicable, because they use spatial indices for low-dimensional spaces. Approximate k NN algorithms are in need to cope with the high processing costs in high-dimensional spaces.

Cross-References

- [Indexing](#)
- [Query Processing: Computational Geometry](#)
- [Query Processing: Joins](#)

References

- Aji A, Wang F, Saltz JH (2012) Towards building a high performance spatial query system for large scale medical imaging data. In: Proceedings of the 20th international conference on advances in geographic information systems (SIGSPATIAL), pp 309–318
- Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) Aqwa: adaptive query workload aware partitioning of big spatial data. Proc VLDB Endow 8(13):2062–2073
- Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517

- Cahsai A, Ntarmos N, Anagnostopoulos C, Triantafillou P (2017) Scaling k-nearest neighbours queries (the right way). In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS), pp 1419–1430
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Eldawy A, Mokbel MF (2013) A demonstration of spatial-hadoop: an efficient mapreduce framework for spatial data. *Proc VLDB Endow* 6(12):1230–1233
- Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. *Acta Informatica* 4(1):1–9
- Ghemawat S, Gobioff H, Leung ST (2003) The google file system. In: Proceedings of the nineteenth ACM symposium on operating systems principles (SOSP), pp 29–43
- Gu Y, Liu G, Qi J, Xu H, Yu G, Zhang R (2016) The moving K diversified nearest neighbor query. *IEEE Trans Knowl Data Eng* 28(10):2778–2792
- Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data (SIGMOD), pp 47–57
- Han D, Stroulia E (2013) Hgrid: a data model for large geospatial data sets in hbase. In: 2013 IEEE 6th international conference on cloud computing (CLOUD), pp 910–917
- Hjaltason GR, Samet H (1995) Ranking in spatial databases. In: Proceedings of the 4th international symposium on advances in spatial databases (SSD), pp 83–95
- Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC (2012) Key formulation schemes for spatial index in cloud data managements. In: 2012 IEEE 13th international conference on mobile data management (MDM), pp 21–26
- Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) idistance: an adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans Database Syst* 30(2):364–397
- Koudas N, Ooi BC, Tan KL, Zhang R (2004) Approximate nn queries on streams with guaranteed error/performance bounds. In: Proceedings of the thirtieth international conference on very large data bases (VLDB), vol 30, pp 804–815
- Lawder JK, King PJH (2001) Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Rec* 30(1):19–24
- Leutenegger ST, Lopez MA, Edgington J (1997) Str: a simple and efficient algorithm for r-tree packing. In: Proceedings 13th international conference on data engineering (ICDE), pp 497–506
- Li C, Gu Y, Qi J, Yu G, Zhang R, Yi W (2014) Processing moving kNN queries using influential neighbor sets. *Proc VLDB Endow* 8(2):113–124
- Li C, Gu Y, Qi J, Yu G, Zhang R, Deng Q (2016) INSQ: an influential neighbor set based moving kNN query processing system. In: Proceedings of the 32nd IEEE international conference on data engineering (ICDE), pp 1338–1341
- Nishimura S, Das S, Agrawal D, Abbadi AE (2011) Mdhbase: a scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 2011 IEEE 12th international conference on mobile data management (MDM), vol 01, pp 7–16
- Nutanong S, Zhang R, Tanin E, Kulik L (2008) The v*-diagram: a query-dependent approach to moving kNN queries. *Proc VLDB Endow* 1(1): 1095–1106
- Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on principles of database systems (PODS), pp 181–190
- Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD), pp 71–79
- Sellis TK, Roussopoulos N, Faloutsos C (1987) The r+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of the 13th international conference on very large data bases (VLDB), pp 507–518
- Wang Y, Zhang R, Xu C, Qi J, Gu Y, Yu G (2014) Continuous visible k nearest neighbor query on moving objects. *Inf Syst* 44:1–21
- Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: efficient in-memory spatial analytics. In: Proceedings of the 2016 SIGMOD international conference on management of data (SIGMOD), pp 1071–1085
- Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems (SIGSPATIAL), pp 70: 1–70:4

Query Processing: Computational Geometry

Amr Magdy

University of California, Riverside, California,
USA

Definitions

Computational geometry studies computational algorithms on computer objects that are modeled as geometric shapes, e.g., points, lines, polygons, and surfaces.

Overview

Geometry has been used for decades to model computer objects with geometric shapes in two- and three-dimensional spaces, e.g., points, lines, polygons, and surfaces. Naturally, different applications that use these objects need to develop algorithms that process and query the geometric shapes for different scenarios and operations. This has started the evolution of computational geometry field that focuses on designing efficient algorithms for different operations and queries on geometric objects. These algorithms are used in several application domains such as computer graphics, computer-aided design and manufacturing (CAD/CAM), geographic information systems (GIS), and robotics motion planning and visibility problems. Part of these application domains intersect with the big data literature due to the large volume of data and/or the large frequency of operations, depending on the application. For example, OpenStreetMap geographic data contains over 6 billion objects, while computer graphics applications perform a plethora of operations every second. Such high-intensive computation necessitates efficient and scalable algorithms for computational geometry operations and queries. In this article, we highlight the main contributions of the research community in the field and give pointers for more comprehensive readings and future directions.

The existing big data literature focus on a set of computational geometry query operations in the two-dimensional (2D) space. 2D computational geometry is mainly abstracting different applications into query operations on three basic geometric objects: points, line segments, and polygons (Fig. 1a). Polygons can be regular shaped, e.g., rectangular search range, or arbitrarily shaped, e.g., country boundaries. In this section, we define common 13 query operations on 2D geometric objects that are used in different applications (de Berg et al. 2008; Lee and Preparata 1984). These operations can be categorized into four types: point operations, line operations, polygon operations, and hybrid operations. All operations assume Euclidean space and Euclidean distance.

(1) **Point query operations:** these operations are performed on a set of input points, while the output can be either a set of points or polygons.

Definition 1 (Closest pair) Given a set of input points P , the *closest pair* operation finds two points $\{p_1, p_2\} \subseteq P$ so that $distance(p_1, p_2) \leq distance(p_i, p_j), \forall p_i, p_j \in P, i \neq j$. So, p_1 and p_2 are the closest pair of points in P out all possible pairs without repetition.

Definition 2 (Farthest pair) Given a set of input points P , the *farthest pair* operation finds two points $\{p_1, p_2\} \subseteq P$ so that $distance(p_1, p_2) \geq distance(p_i, p_j), \forall p_i, p_j \in P, i \neq j$. So, p_1 and p_2 are the farthest pair of points in P out all possible pairs without repetition.

Definition 3 (Convex hull) Given a set of input points P , the *convex hull* operation finds a subset of P that contains vertices of the smallest convex polygon that contains *all* points in P . Convex polygons have all interior angles measuring less than 180° .

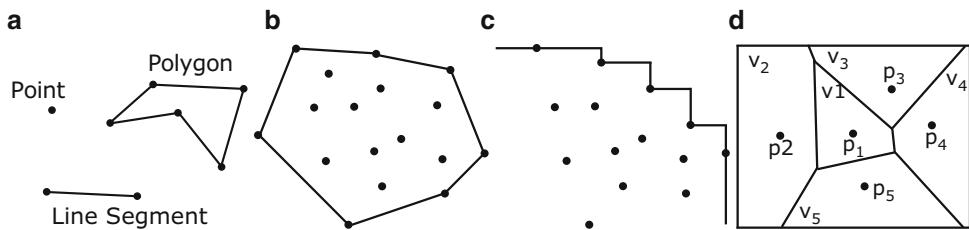
Figure 1b shows an example of the convex hull operation output. The operation output is the set of points on the polygon perimeter ordered in a clockwise direction.

Q

Definition 4 (Skyline) Given a set of input points P , the *skyline* operation finds a subset $P_d \subseteq P$, where $P_d = \{p_i \in P, p_i \text{ is not dominated by } p_j, \forall p_j \in P, i \neq j\}$. A point p_i dominates point p_j if and only if $p_i.x \geq p_j.x$ and $p_i.y \geq p_j.y$, where x and y are the 2D coordinates in the Euclidean space.

Figure 1c shows an example of the skyline operation output. The output is the set of points that represent vertices of the line segments in the figure.

Definition 5 (Voronoi diagram) Given a set of input points P , the *Voronoi diagram* operation finds a set of Voronoi partitions V , such



Query Processing: Computational Geometry, Fig. 1 Geometric basic objects and point query operations. (a) Geometric objects. (b) Convex hull. (c) Skyline. (d) Voronoi diagram

that $|V| = |P|$ and each partition v_i has a distinct seed point $p_i \in P$, $1 \leq i \leq |P|$. A Voronoi partition v_i with a seed point p_i is an area in the Euclidean space R^2 where $v_i = \{p_x \in R^2, \text{distance}(p_x, p_i) \leq \text{distance}(p_x, p_j), \forall p_j \in P, i \neq j\}$. This makes each Voronoi partition v_i a union of all points in the Euclidean space where p_i is the nearest point among all points in P .

Figure 1d shows an example of a Voronoi diagram for a set of five points $P = \{p_1, p_2, p_3, p_4, p_5\}$. The figure shows corresponding five Voronoi partitions $V = \{v_1, v_2, v_3, v_4, v_5\}$. All points in the partition v_1 , for example, have p_1 as their nearest neighbor among all points in P and similarly v_2 with p_2 and so on.

(2) **Line query operations:** this operation is performed on a set of input line segments, while the output is either a Boolean value (true or false) or a set of line segments.

Definition 6 (Line segment intersection) Given two input line segments L_1 and L_2 , the *line segment intersection* operation finds whether or not L_1 and L_2 intersect.

(3) **Polygon query operations:** these operations are performed on a set of input polygons, while the output is another set of polygons.

Definition 7 (Polygon intersection) Given two input polygons A and B , the *polygon intersection* operation $A \cap B$ finds a polygon that represents the common area of A and B .

Figure 2a shows an example of polygon intersection operation. The gray-shaded area shows the output intersection polygon.

Definition 8 (Polygon union) Given two input polygons A and B , the *polygon union* operation $A \cup B$ finds a polygon that represents the unity of both A and B areas.

Figure 2c shows an example of polygon union operation. The gray-shaded area shows the output union polygon.

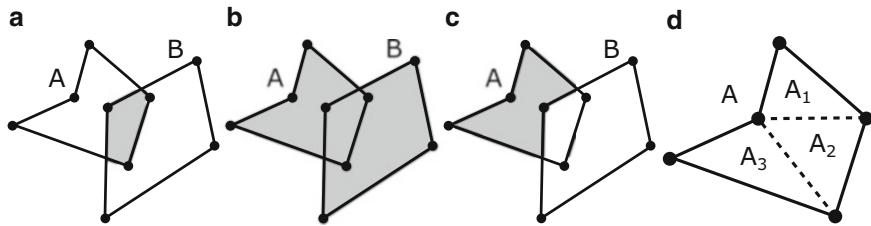
Definition 9 (Polygon difference) Given two input polygons A and B , the *polygon difference* operation $A - B$ finds a polygon that represents the area of A that is not part of B .

Figure 2c shows an example of polygon difference operation. The gray-shaded area shows the output difference polygon.

Definition 10 (Polygon triangulation) Given an input polygon A , the *polygon triangulation* operation finds a set of disjoint triangles $\{A_1, A_2, \dots, A_n\}$ such that $A = A_1 \cup A_2 \cup \dots \cup A_n$.

Figure 2d shows an example of polygon triangulation operation. The polygon A is decomposed into three triangles: A_1 , A_2 , and A_3 . The output triangles are disjoint and their union covers the whole area of polygon A .

(4) **Hybrid query operations:** these operations are performed on an input hybrid set of points, line segments, and/or polygons, while



Query Processing: Computational Geometry, Fig. 2 Polygon query operations. (a) Intersection. (b) Union. (c) Difference. (d) Triangulation

the output is either Boolean true or false or another hybrid set of points, line segments, and/or polygons.

Definition 11 (Point in polygon) Given a point p and a polygon A , the *point in polygon* operation finds whether or not p lies inside the perimeter of A .

Definition 12 (Range search) Given a set of points P and a rectangular polygon R , the *range search* operation finds a set of points $P_R \subseteq P$ such that $P_R = \{p_i \in P, p_i \text{ lies inside } R\}$.

Definition 13 (Simplex range search) Given a set of points P and a 2D simplex polygon, i.e., triangle, T , the *simplex range search* operation finds a set of points $P_T \subseteq P$ such that $P_T = \{p_i \in P, p_i \text{ lies inside } T\}$. In 2D, this operation can be referred as *triangular range search* because a 2D simplex is a triangle.

Key Research Findings

This section gives a brief overview on traditional computational geometry structures and query processing. Then, it highlights query processing techniques on big geometric data in modern big data systems.

Traditional Query Processing

This section highlights the major families of algorithms and data structures that are traditionally employed to solve and speed up computational

geometry query operations in different applications.

(1) **Plane sweep algorithms** (or *sweep line algorithms*): A family of algorithms that use a conceptual sweep line to solve various problems in Euclidean space. The main idea is to imagine a straight line, mostly a horizontal or a vertical line, moved across the plane and stops at certain discrete points that correspond to the input set of points or line segment vertices. Then, geometric operations are performed on geometric objects that either intersect or are in the immediate vicinity of the sweep line whenever it stops. The final solution is available once the line has passed over all the input objects. This family has a variety of algorithms to solve points closest pairs (Bartling and Hinrichs 1992), polygon intersections (Nierergelt and Preparata 1982), convex hull (Andrew 1979), and Voronoi diagrams (Fortune 1987).

(2) **Multidimensional trees**: A family of tree data structures that are used to index multidimensional data, e.g., 2D or 3D, including points, line segments, and polygons. They are used to support efficient range search query operation. Three major tree structures in this family are kd-trees, quadtrees, octrees, and their different variations. Kd-tree (Bentley 1975) is a binary tree that index k-dimensional data points. Every non-leaf node is dividing the space into two halves through a hyperplane perpendicular to one of the k-dimensions. Points to the left of this hyperplane are indexed by the left subtree of that node and points right of the hyperplane are indexed by the right subtree. For example, if for a particular node splits the x-axis, all points in the subtree with a smaller x value will appear in the left subtree and all points with larger x value

will be in the right subtree. Quadtree (Samet 1984) is a more generic structure and has several variations that can index points, line segments, and/or polygons. A quadtree is a four-ary tree and consists of multiple levels, each level covers the whole indexed space, and level i has 4^i nodes. Level 0 consists of one (4^0) root node that covers the whole space. Then, at level 1 the root node is divided into four children nodes of equal areas, and each of the children nodes are divided into four equal nodes at level 2 and so on. Each tree node has a maximum capacity of m objects, so a node is divided only if has more than m objects. Quadtree is mainly indexing two-dimensional data. Octree (Meagher 1982) is the corresponding three-dimensional version where each node is divided into eight children nodes in the three-dimensional space. Both quadtrees and octrees are used heavily in different operations for computer graphics applications. In addition, quadtrees are used to index and query location data in geographic information systems and location-based services.

(3) **Point in polygon:** A point in polygon query can be processed through two algorithms: *crossing number* and *winding number* algorithms. The crossing number algorithm, also known as ray casting, counts the number of intersections of the query polygon sides with an infinite line starts from the query point and continues to infinity. If the crossing number is even, then the point is outside the polygon; otherwise, it is inside. This algorithm, however, is considering only simple polygon where there is no self-intersection among the polygon sides and might fail in complex polygons. To overcome this, the winding number algorithm, as its name suggests, calculates the winding number (a known concept in mathematics) of the polygon around the point. If this number is zero, then the point is outside the polygon; otherwise, it is inside. The winding number is mainly counting the number of cycles the polygon sides are winding around the point. Calculating this number is performed through summation of angles between the query point and end points of the query polygon sides. Then, the winding number is the nearest multiple of 360° .

(4) **Line segment intersection:** It is straightforward to efficiently find intersection of two line segments based on geometry laws. However, if a large number of line segments are involved in the query, the query becomes increasingly expensive. This query is processed by a sweep line technique where the algorithm tracks which line segments are intersecting with the sweep vertical line at each point along the axis using a dynamic data structure based on binary search trees. This phase eliminates the majority of unnecessary checks on segment pairs. Then, a significantly smaller number of checks are performed on segment pairs that are potentially intersecting.

(5) **Skyline:** Skyline has a variety of query processing techniques and variations (Kalyvas and Tzouramanis 2017). Several divide and conquer algorithms have been proposed to tackle this query. The algorithm is dividing the input data into smaller subsets, so each subset fits in main memory. Then, the skyline of each subset is computed efficiently. Finally, the global skyline is computed by merging local skylines of each subset. One way of efficient skyline computation is using multidimensional trees, e.g., R-trees or quadtrees, to early prune points that do not contribute to the final skyline and eliminate redundant dominance checks.

Computational Geometry on Big Data

Scaling computational geometry operations on big volume data has been developed over two main stages that overlap in time, both exploiting parallel computing environments to speed up computational geometry operations. The first stage is exploiting multicore architectures (Akl and Lyons 1993; Aggarwal et al. 1988; Liknes et al. 2014; Miller and Stout 1988; McKenney et al. 2017) with shared-memory parallel computation model, while the second stage is exploiting distributed big data systems (Eldawy et al. 2013; Puri et al. 2017; Vlachou et al. 2008; Zhang et al. 2016) with shared-nothing computation model. This section highlights query processing techniques in both stages.

Geometric query processing in multicore environments. Multicore environments have been

exploited in parallel query processing techniques to support efficient computational geometry operations on large datasets that fit in main memory. These techniques have adapted a generic framework that work in three stages: (a) divide the input data into smaller subsets, (b) compute the geometric operation on each subset in parallel efficiently, and (c) combine the results of the subsets to get the final output on the whole dataset. The details of each stage depend on the geometric operation and the optimizations of the underlying algorithm. We elaborate on this framework through two examples of computing *convex hull* and *closest pair* in parallel multicore environments. It is noteworthy that the presented examples are not the only parallel algorithms for these operations; however, they show instances of the described generic framework.

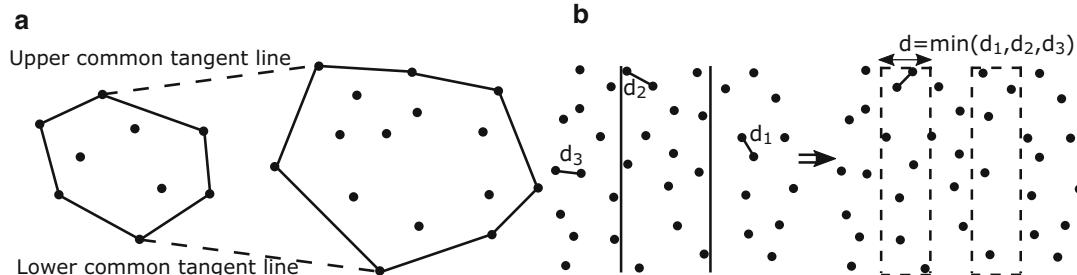
Example 1 (Convex Hull) One way to compute the convex hull of a set of points through the above framework is to divide the input data into smaller subsets that are linearly separable. Then, the convex hull on each subset is computed and combined to get the convex hull of bigger sets out of smaller ones through upper and lower common tangent lines. Figure 3a shows an example of merging convex hull polygons of two sets of points to get the convex hull polygon of the combined set. Given the two polygons, the algorithm finds two line segments, namely, upper common tangent line and lower common tangent line, as depicted in the figure. These two lines represent upper and lower envelopes for all points on the two convex polygons. Then, the algorithm

removes all points between these two lines and keeps the other points as the final convex hull.

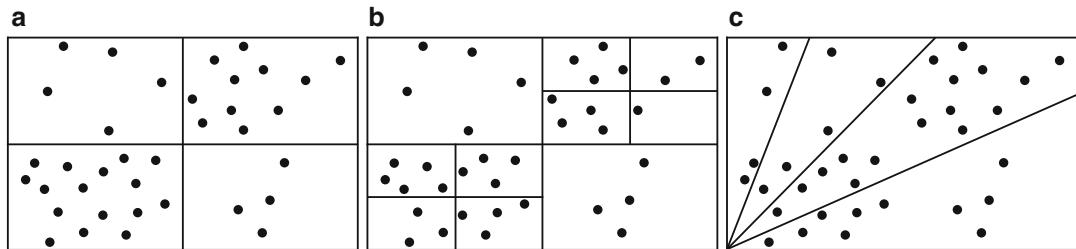
Example 2 (Closest Pair) Figure 3b shows an example of a parallel closest pair algorithm. The algorithm divides the input points into linearly separable subsets S_1 , S_2 , S_3 and computes the local closest pair in each subset with corresponding closest distances d_1 , d_2 , and d_3 , respectively. Then, to compute the global closest pair, a buffer zone of width d is considered around the partition lines of the three subsets, where $d = \min(d_1, d_2, d_3)$. The points in this buffer zone are checked for the closest points to ensure getting the global closest pair.

In addition to convex hull and closest pair, parallel algorithms in multicore environments exist for different geometric operations including polygon triangulation, Voronoi diagrams, line segment intersections, minimizing a circumscribing triangle, and recursive data structures for three-dimensional queries (Aggarwal et al. 1988; Akl and Lyons 1993).

Geometric query processing in big data systems. With the increasing volume of datasets, fitting data in main memory is no longer a possibility for several applications. For example, OpenStreetMap geographic data contains over 6 billion objects, which need to be processed in disk-based environments to accommodate such large volumes. For that, new techniques have been developed to exploit the recent advances in distributed big data systems such as MapReduce



Query Processing: Computational Geometry, Fig. 3 Examples of parallel operations in multicore environments. (a) Merging convex hulls. (b) Parallel closest pair



Query Processing: Computational Geometry, Fig. 4 Examples of different partitioning techniques. (a) Fixed grid partitioning. (b) Dynamic grid partitioning. (c) Angle partitioning

environments. These techniques mainly focus on batch analysis applications, e.g., geospatial analysis, rather than interactive querying applications, e.g., robot motion planning. Therefore, batch analysis environments such as MapReduce are suitable incubator to realize them.

Distributed geometric techniques in big data systems adopt the same three-stage generic framework that has been employed in multicore environments: divide input data, compute geometric operation locally, and merge local outputs to get the global one. However, the details of each stage are different due to the different computation environment and therefore the different performance optimization goals. For example, MapReduce-based techniques optimize for lower I/O and lower network communication, which were not the optimization goals in main memory environments. The main differences in MapReduce techniques compared to traditional parallel techniques come in two folds. First, the early focus of traditional techniques is to optimize the theoretical runtime complexity with later focus on experimenting actual runtime of the algorithms. On the contrary, MapReduce techniques mainly focus on optimizing the actual runtime with almost no focus on the theoretic bounds. Second, MapReduce techniques have different filtering and partitioning techniques that optimize for I/O and network communication costs in shared-nothing architectures. Figure 4 shows three examples of different partitioning techniques: fixed grid partitioning, dynamic grid partitioning, and angle-based partitioning. The pros and cons of each partitioning technique are studied in the literature.

Examples of Application

Computational geometry algorithms are used in several application domains such as computer graphics, computer-aided design and manufacturing (CAD/CAM), geographic information systems (GIS), and robotics motion planning and visibility problems (de Berg et al. 2008; Lee and Preparata 1984). For example, in GIS applications, 2D computational geometry structures and algorithms are heavily used for range search queries, point in polygon operations, and nearest neighbor search.

Future Directions for Research

The presented techniques in this entry have not exploited the recent advances in both hardware technologies, e.g., GPUs, and software systems, e.g., Apache Spark and Apache Flink. The advances in GPU technology have recently attracted emerging efforts to exploit them in speeding up operations and applications in the geometric space (Chen and Chen 2016; Sebastian et al. 2014). However, these efforts form few steps in a promising research direction that might be expanded to fill more gaps. One of such gaps is the lack of holistic systems that gather the scattered techniques and exploit GPU infrastructures to develop end-to-end solutions. In addition to the new hardware, the recent advances in software systems have not yet exploited in the field of computational geometry. The current literature depends on Hadoop MapReduce as the underlying distributed framework to handle disk-

resident big data. However, the new in-memory processing frameworks such as Apache Spark and Apache Flink have not been exploited, yet, they provide much better interactive response compared to Hadoop-based frameworks.

Cross-References

- ▶ [GPU-based Hardware Platforms](#)
- ▶ [Indexing](#)
- ▶ [Parallel Processing with Big Data](#)
- ▶ [Query Processing – kNN](#)

References

- Aggarwal A, Chazelle B, Guibas LJ, Ó'Dúnlaing C, Yap C (1988) Parallel computational geometry. *Algorithmica* 3:293–327
- Akl SG, Lyons KA (1993) Parallel computational geometry. Prentice Hall, Englewood Cliffs
- Andrew AM (1979) Another efficient algorithm for convex hulls in two dimensions. *Inf Process Lett* 9(5): 216–219
- Bartling F, Hinrichs KH (1992) A plane-sweep algorithm for finding a closest pair among convex planar objects. In: Proceedings of the annual symposium on theoretical aspects of computer science, STACS, pp 221–232
- Bentley JL (1975) Multidimensional binary search trees used for associative searching. *Commun ACM* 18(9):509–517
- Chen H, Chen W (2016) GPU-accelerated blind and robust 3D mesh watermarking by geometry image. *Int J Multimedia Tools Appl* 75(16):10077–10096
- de Berg M, Cheong O, van Kreveld MJ, Overmars MH (2008) Computational geometry: algorithms and applications, 3rd edn. Springer, Berlin/Heidelberg
- Eldawy A, Li Y, Mokbel MF, Janardan R (2013) CG_Hadoop: computational geometry in mapreduce. In: Proceedings of the international conference on advances in geographic information systems, ACM SIGSPATIAL, pp 284–293
- Fortune S (1987) A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2:153–174
- Kalyvas C, Tzouramanis T (2017) A survey of skyline query processing. CoRR abs/1704.01788. <http://arxiv.org/abs/1704.01788>
- Lee DT, Preparata FP (1984) Computational geometry – a survey. *IEEE Trans Comput* 33(12):1072–1101
- Liknes S, Vlachou A, Doulkeridis C, Nørvåg K (2014) APSkyline: improved skyline computation for multicore architectures. In: Proceedings of the international conference on database systems for advanced applications, DASFAA, pp 312–326
- McKenney M, Frye R, Dellamano M, Anderson K, Harris J (2017) Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. *GeoInformatica* 21(1):151–174
- Meagher D (1982) Geometric modeling using cctree encoding. *Comput Graph Image Process* 19(2):129–147
- Miller R, Stout QF (1988) Efficient parallel convex hull algorithms. *IEEE Trans Comput* 37(12):1605–1618
- Nievergelt J, Preparata FP (1982) Plane-sweep algorithms for intersecting geometric figures. *Commun ACM* 25(10):739–747
- Puri S, Agarwal D, Prasad SK (2017) Polygonal overlay computation on cloud, Hadoop, and MPI. In: Encyclopedia of GIS, pp 1598–1606
- Samet H (1984) The quadtree and related hierarchical data structures. *ACM Comput Surv* 16(2):187–260
- Sebastian J, Sivadasan N, Banerjee R (2014) GPU accelerated three dimensional unstructured geometric multi-grid solver. In: Proceedings of the international conference on high performance computing & simulation, HPCS, pp 9–16
- Vlachou A, Doulkeridis C, Kotidis Y (2008) Angle-based space partitioning for efficient parallel skyline computation. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD, pp 227–238
- Zhang J, Jiang X, Ku W, Qin X (2016) Efficient parallel skyline evaluation using mapreduce. *IEEE Trans Parallel Distributed Syst* 27(7):1996–2009

Query Processing: Joins

Nikos Mamoulis

Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

Q

Synonyms

[Parallel spatial joins](#); [Spatial joins](#); [Spatial joins in clusters](#)

Definitions

Given two collections R and S of spatial objects and a spatial predicate θ , the spatial join computes the pairs of objects (r, s) such that $r \in R$, $s \in S$, and $r \theta s$. Common spatial predicates are intersect, inside, and contains. For example, con-

sider a GIS application, where R is a collection of rivers and S is a collection of road segments. The spatial join of R and S finds the pairs of rivers and roads that intersect. If the spatial objects are points, the most common type of spatial join is the distance join, where θ is “*within distance ϵ from*”; here, ϵ is a given threshold. For example, assuming that R is the set of hotels on a city map and S is the set of restaurants on the same map, the distance join finds pairs of hotels and restaurants that are sufficiently close to each other (e.g., $\epsilon = 100$ m).

Overview

For objects with spatial extent, a common practice is to evaluate spatial joins (and queries in general) in two steps. During the *filter* step, the *minimum bounding rectangles* (MBRs) of the objects are compared, and if they do not satisfy θ , they are filtered out from consideration. During the refinement step, for each object pair that passes the filter step, θ is tested using the exact geometries of the objects.

A wide variety of spatial join algorithms have been proposed in the literature (Jacox and Samet 2007; Mamoulis 2009). Single-threaded spatial join techniques were first proposed for memory-resident data. Given the quadratic cost of a naive nested-loops approach, efforts were put into developing methods based on plane sweep, after sorting the data in one dimension (Preparata and Shamos 1985; Arge et al. 1998). These algorithms were later used as modules for joining larger datasets that do not fit in memory. In this case, the typical procedure is to partition one or both datasets with the help of a data structure, e.g., a spatial access method (Samet 1990), and then execute the join as a set of (independent) joins between partitions which can be accommodated in memory. Objects may already be indexed before join processing, as spatial indexes are also useful for other spatial queries, such as range selections and nearest neighbor search. As a result, spatial join algorithms have been proposed for both cases of non-indexed and indexed inputs.

Partitioning-based approaches may also improve the performance of in-memory joins, if the datasets are very large, in a similar manner as relational hash join algorithms. In the late 1990s, the research interest gradually shifted to parallel processing of spatial joins for shared-nothing and shared-memory architectures (Brinkhoff et al. 1996; Patel and DeWitt 2000). The developed approaches attempt to divide the problem into multiple independent smaller problems of similar cost, which are then processed in parallel. Finally, since the late 2000s, the focus is on processing of spatial joins in a distributed environment using the MapReduce or Spark frameworks (Eldawy and Mokbel 2015; Zhang et al. 2009; Xie et al. 2016) and on in-memory joins (Nobari et al. 2013).

Key Research Findings

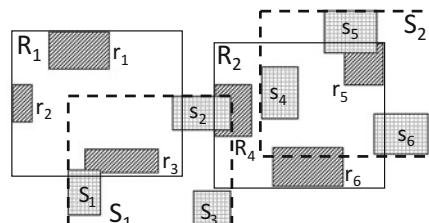
We will focus on the filter step of spatial intersection joins. Hence, the join inputs are considered to be two collections R and S of rectangles, and the objective is to find pairs (r, s) , $r \in R, s \in S$, such that r intersects s (i.e., r and s have at least one common point). Distance joins can be modeled as intersection joins, if we extend the objects in one of the inputs by ϵ in all dimensions and directions (Nobari et al. 2013).

For in-memory spatial join processing, plane sweep (Preparata and Shamos 1985) is a well-studied technique. The most commonly used version of plane sweep is suggested by Brinkhoff et al. (1993). The inputs R and S are sorted based on their lowest value in one dimension (e.g., the x -dimension). Then, the inputs are concurrently scanned (as in a merge join), and each encountered value can be considered as a stop position of a line that sweeps along the sorting dimension. For every encountered value (e.g., a lower x -endpoint $r.xl$ of a rectangle $r \in R$), the other collection (e.g., S) is *forward scanned* until an $s \in S$ whose $s.xl$ is *after* the upper x -endpoint $r.xu$ of r . All $s \in S$ found in the scan are guaranteed to x -intersect r , and a y -intersection test is performed on-the-fly. Other versions of plane sweep, based on maintenance of *active lists*

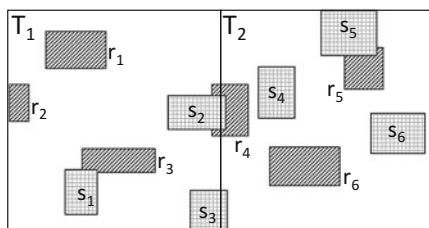
at every position of the sweep line, were studied in Arge et al. (1998).

Big spatial data may not fit in memory, and even if they do, it would be expensive to spatially join them using plane sweep. Given this, *data partitioning* has been used as a technique to break down the problem into numerous small problems each of which can be solved fast in memory. In a nutshell, objects are divided to groups based on their spatial proximity. A partition from R should then be joined with a partition from S if their MBRs intersect. Spatial join techniques which were proposed in the 1990s apply this approach. They can be classified into *single-assignment*, *multi-join* (SAMJ) methods or *multi-assignment*, *single-join* (MASJ) approaches (Lo and Ravishankar 1996). SAMJ methods partition the two datasets differently so that each object is assigned to exactly one partition, but a partition from R may have to be joined with multiple partitions from S . MASJ, on the other hand, may assign objects to multiple partitions, but each partition from R is joined with exactly one partition from S (which has exactly the same MBR). Figure 1a illustrates an example of SAMJ. The dark-gray rectangles belong to R , and they are partitioned to two groups R_1 and R_2 , while the light-gray rectangles belong to S , and they are divided into S_1 and S_2 . Group R_1 only needs to be joined with S_1 because its MBR is disjoint with that of S_2 . However, R_2 should be joined with both S_1 and S_2 . In Fig. 1b, MASJ is applied. The datasets are again partitioned to tiles T_1 and T_2 , and the data from R in each tile only have to be joined with the group of S in the same tile. However, objects $\{r_4, s_2, s_3\}$, which cross the borders of the two tiles, are replicated in both of them.

SAMJ is naturally applied when the two datasets are spatially indexed before the join by a separate spatial access method (e.g., a different R-tree for each input). The *R-tree join* (RJ) algorithm (Brinkhoff et al. 1993) takes as input two R-trees, which index R and S , respectively. RJ starts by finding all pairs of entries (e_R, e_S) one from each root node of the trees that intersect. For each such pairs, it recursively applies the same procedure for the nodes pointed by e_R and e_S , until pairs of leaf node entries (which



(a) multi-assignment, single-join (MASJ)



(b) single-assignment, multi-join (SAMJ)

Query Processing: Joins, Fig. 1 Two classes of partitioning techniques. (a) Multi-assignment, single-join (MASJ). (b) Single-assignment, multi-join (SAMJ)

correspond to intersecting object MBRs) are found. For example, if R_1 and R_2 (S_1 and S_2 , respectively) in Fig. 1a are the two children of an R-tree root that indexes R (S , respectively), then RJ would use their MBRs to determine that R_1 only needs to be joined with S_1 . To join each pair of nodes, plane sweep is used. RJ was later extended to a multiway join processing algorithm (Mamoulis and Papadias 2001) that applies to multiple R-trees.

One of the first MASJ approaches is *spatial hash join* (Lo and Ravishankar 1996). First, input R is read and partitioned to buckets, which are expected to fit in memory. The bucket extents are determined by sampling. The objects are divided to partitions that may spatially overlap each other, but each object goes into exactly one bucket. Dataset S is then partitioned to buckets that have the same extent as those for R , but an object from S is replicated to all buckets whose spatial extent overlaps. A 1-1 bucket-to-bucket join is then performed. The most popular MASJ approach is *partition-based spatial-merge join* (PBSM) (Patel and DeWitt 1996). PBSM divides the space by a regular grid, and objects from both join inputs are assigned to all tiles which spatially

overlap them. For example, in Fig. 1b, T_1 and T_2 could be tiles in a large rectangular grid that can be used to partition both R and S . We may have a huge number of tiles which have an uneven number of objects. In order to make the partitions as balanced as possible, we choose the number of tiles to be much larger than the number of partitions and then assign multiple tiles to each partition, such that the expected set of objects per partition fits in memory and the partitions are as balanced as possible. The assignment of tiles to partitions can be done in round-robin or by using a hash function. For example, assuming three partitions, tiles T_1, T_4, T_7 , etc. are assigned to partition 1; T_2, T_5, T_8 , etc. to partition 2; etc. In the join phase, for each partition, PBSM accesses the objects from R and the objects from S and performs their join in memory (e.g., using plane sweep). Since two replicated objects may intersect in multiple tiles (e.g., see r_4 and s_2 in Fig. 1b), duplicate results may be produced. In order to avoid duplicates, a join result is reported by a tile only if a prespecified reference point (e.g., the top-left corner) of the intersection region is in the tile (Dittrich and Seeger 2000).

Size separation spatial join (Koudas and Sevcik 1997) is a SAMJ algorithm that applies a hierarchical grid decomposition and avoids object replication. Objects are assigned to the smallest tile in the hierarchy that entirely contains them. All pairs of tiles that have an ancestor-descendant relation in the hierarchy are joined to each other. *Scalable sweeping-based spatial join* (Arge et al. 1998) divides the space to stripes according to one dimension and applies plane sweep at each stripe in the direction of the other dimension.

Join algorithms that apply on one indexed input (by an R-tree) take advantage of the index to guide the partitioning of the second (raw) input. One approach is to build a second R-tree using the existing tree to guide the MBRs of the top tree levels (Lo and Ravishankar 1998). Another approach is to use the existing R-tree to create bucket extents for hashing the raw input (Mamoulis and Papadias 2003) and then apply a hash join (Lo and Ravishankar 1996).

Early efforts on parallelizing spatial joins include extensions of the R-tree join and PBSM

algorithms in a distributed environment of single-core processors with local storage. Consider the case of joining two R-trees. Since overlapping pairs of root entries define independent join processes for the corresponding subtrees, these tasks can be assigned to different processors (Brinkhoff et al. 1996). Two tasks may access a common subtree; therefore, a virtual global buffer is shared among processors to avoid accessing the same data twice from the disk. An early approach in parallelizing PBSM (Zhou et al. 1997) asks processors to perform the partitioning of data to tiles independently and in parallel. Then, each processor is assigned one partition and processors exchange data, so that each one gets all objects that fall in its partition. The join phase is finally performed in parallel. A *partial spatial surrogate* approach (Patel and DeWitt 2000) replicates MBRs instead of entire tuples, while the exact object geometry is assigned to a single (home) node. In the join phase, each operator looks at the fragments of the declustered datasets residing on its local disks and joins them using any centralized spatial join algorithm.

Recently, the research interest shifted to spatial join processing for distributed cloud systems and multi-core processors. The popularity and commercial success of the MapReduce framework motivated the development of spatial join processing algorithms on clusters. The *spatial join with MapReduce* (SJMP) algorithm (Zhang et al. 2009) is an adaptation of the PBSM algorithm in this direction. Initially, the space is divided by a grid, and tiles are grouped to partitions in a round-robin fashion after considering them in a space-filling curve order. During the *map* phase, each object is assigned to one or more partitions based on the tiles it overlaps. Each partitioned object also carries the set of tiles it intersects. Each partition corresponds to a *reduce* task. The reducers perform their joins by dividing the space that corresponds to them into stripes and performing plane sweep for each tile. Duplicate results are avoided by reporting a join pair only at the tile with the smallest id where the two objects commonly appear. In the *Hadoop-GIS* system (Aji et al. 2013), spatial joins are computed in a similar fashion. The space

is divided by a grid to tiles, and the objects in each tile are stored locally at nodes in the HDFS. A *global index* that is shared between nodes is used to find the HDFS files where the data of each tile are located. The local data at each node are also locally indexed. Local joins are performed by each node separately, and the results are merged. The *SpatialHadoop* system (Eldawy and Mokbel 2015) follows a similar approach where a global index per dataset is stored at a Master node; however, different datasets may have different partitioning that could preexist before queries. If the two join inputs are partitioned differently, then two options exist: (1) use existing partitions and perform joins for every pair of partitions that overlap (they could be many) or (2) repartition the smaller file using the same partition boundaries as the larger file and apply MASJ. The cost of each of the two options is estimated, and the cheapest one is selected accordingly. Implementations of spatial joins using Spark, where the data, indexes, and intermediate results are shared in the memories of all nodes in a cluster, have also been proposed (You et al. 2015; Yu et al. 2015; Xie et al. 2016), with a focus on effective spatial indexing of the *resilient distributed datasets* (RDDs) that are generated during the process. Processing spatial joins in parallel with a focus on minimizing the cost of the refinement step was studied by Ray et al. (2014). During data partitioning, the exact geometries of objects are clipped and distributed to partitions (that may be handled by different nodes) in order to avoid any communication between nodes when object pairs whose MBRs intersect are refined.

In-memory joins were recently reconsidered, given that memories have become large and that applying plane sweep for big data inputs is too slow due to the large number of objects intersected by every position of the sweep line. Partitioning-based approaches, such as PBSM, can be used to break the problem into numerous small instances that can be solved fast (i.e., a divide and conquer approach). Algorithm *TOUCH* (Nobari et al. 2013) is a recent effort in this direction, designed for scientific applications that join huge datasets that have different density and skew. *TOUCH* first bulk-loads an R-tree for one

of the inputs using the STR technique (Leutenegger et al. 1997). Then, all objects from the second input are assigned to buckets corresponding to the non-leaf nodes of the tree. Each object is hashed to the lowest tree node, whose MBR overlaps it, but no other nodes at the same tree level do. Finally, each bucket is joined with the subtree rooted at the corresponding node with the help of a dynamically created grid data structure for the subtree. A comparison of spatial join algorithms for in-memory data (Nobari et al. 2017) shows that PBSM and *TOUCH* perform best and that the join cost depends on the data density and distribution.

Examples of Application

The amount of spatial data that are collected and need to be analyzed has exploded in the recent years, due to the wide use of mobile and sensing devices and the drop of equipment and telecommunication costs. The spatial join is a core operation in *spatial data management systems* (Güting 1994) and *geographic information systems* (GIS), which manage big amounts of spatial data. In GIS, for example, for a given map, we typically store different thematic layers (e.g., hydrography, road network) which may have to be joined in order to find pairs of elements (e.g., rivers and roads) that intersect.

Spatial joins are especially useful in scientific data analysis. For example, the *Sloan Digital Sky Survey* (Szalay et al. 2000) is a project that collects and analyzes multiple terabytes of sky data. Among the most useful operations for analyzing these data is the spatial join, which can be used as a component for clustering and other higher-level analysis tasks.

Spatial joins also find use in medical applications. For example, they can be used to facilitate the analysis of high-resolution images of human tissue specimens, which can lead to more effective diagnosis, prediction, and treatment of diseases (Aji et al. 2012). Another medical application of spatial joins is to determine neuron synapses in large spatial brain models, i.e., pairs

of neuron branches that are within very small distance to each other (Nobari et al. 2013).

Future Directions for Research

The refinement step of a spatial join can be much more expensive than the filter step if the objects have complex spatial extent. However, the focus of research has been on the filter step, and there is little work on optimizing the refinement step (Ray et al. 2014). Hence, there is room for improving the mechanisms for avoiding refinements wherever possible and for optimizing how refinements are implemented and scheduled in a distributed environment. Finally, for some classes of objects such as line segments, it might be more appropriate to directly use their geometric representation instead of approximating them by MBRs and design specialized join algorithms. In a recent work in this direction (McKenney et al. 2017), a multi-core spatial join algorithm based on plane sweep was designed for line segment collections.

Distance and nearest neighbor joins (Zhang et al. 2004) are evaluated by extending and adapting algorithms for intersection joins. Special algorithms that take into consideration the fact that such joins are typically conducted between point sets could be designed.

In most of the applications that manage spatial data, the data can be preprocessed and can easily fit in the memory of a single commodity machine. Therefore, it is questionable whether we need cloud computing algorithms for spatial joins. On the other hand, multi-core parallelism is gaining ground, so a promising research direction is to design good shared-memory parallel algorithms for spatial joins.

Cross-References

- ▶ [Big Data Indexing](#)
- ▶ [Hadoop](#)
- ▶ [Mobile Big Data: Foundations, State of the Art, and Future Directions](#)
- ▶ [Parallel Join Algorithms in MapReduce](#)

- ▶ [Parallel Processing with Big Data](#)
- ▶ [Query Processing: Computational Geometry](#)
- ▶ [Query Processing – kNN](#)
- ▶ [Spatiotemporal Data: Trajectories](#)
- ▶ [Streaming Big Spatial Data](#)

References

- Aji A, Wang F, Saltz JH (2012) Towards building a high performance spatial query system for large scale medical imaging data. In: SIGSPATIAL/GIS, pp 309–318
- Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz JH (2013) Hadoop-GIS: a high performance spatial data warehousing system over mapreduce. PVLDB 6(11):1009–1020
- Arge L, Procopiuc O, Ramaswamy S, Suel T, Vitter JS (1998) Scalable sweeping-based spatial join. In: VLDB, pp 570–581
- Brinkhoff T, Kriegel HP, Seeger B (1993) Efficient processing of spatial joins using r-trees. In: SIGMOD conference, pp 237–246
- Brinkhoff T, Kriegel H, Seeger B (1996) Parallel processing of spatial joins using R-trees. In: ICDE, pp 258–265
- Dittrich J, Seeger B (2000) Data redundancy and duplicate detection in spatial join processing. In: ICDE, pp 535–546
- Eldawy A, Mokbel MF (2015) SpatialHadoop: a mapreduce framework for spatial data. In: ICDE, pp 1352–1363
- Güting RH (1994) An introduction to spatial database systems. VLDB J 3(4):357–399
- Jacox EH, Samet H (2007) Spatial join techniques. ACM Trans Database Syst 32(1):7
- Koudas N, Sevcik KC (1997) Size separation spatial join. In: SIGMOD conference, pp 324–335
- Leutenegger ST, Edgington JM, López MA (1997) STR: a simple and efficient algorithm for r-tree packing. In: ICDE, pp 497–506
- Lo ML, Ravishankar CV (1996) Spatial hash-joins. In: SIGMOD conference, pp 247–258
- Lo ML, Ravishankar CV (1998) The design and implementation of seeded trees: an efficient method for spatial joins. IEEE Trans Knowl Data Eng 10(1):136–152
- Mamoulis N (2009) Spatial join. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer, New York/London, pp 2707–2714
- Mamoulis N, Papadias D (2001) Multiway spatial joins. ACM Trans Database Syst 26(4):424–475
- Mamoulis N, Papadias D (2003) Slot index spatial join. IEEE Trans Knowl Data Eng 15(1):211–231
- McKenney M, Frye R, Dellamano M, Anderson K, Harris J (2017) Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. GeoInformatica 21(1):151–174

- Nobari S, Tauheed F, Heinis T, Karras P, Bressan S, Ailamaki A (2013) TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In: SIGMOD conference, pp 701–712
- Nobari S, Qu Q, Jensen CS (2017) In-memory spatial join: the data matters! In: EDBT, pp 462–465
- Patel JM, DeWitt DJ (1996) Partition based spatial-merge join. In: SIGMOD conference, pp 259–270
- Patel JM, DeWitt DJ (2000) Clone join and shadow join: two parallel spatial join algorithms. In: ACM-GIS, pp 54–61
- Preparata FP, Shamos MI (1985) Computational geometry – an introduction. Springer, New York
- Ray S, Simion B, Brown AD, Johnson R (2014) Skew-resistant parallel in-memory spatial join. In: SSDBM, pp 6:1–6:12
- Samet H (1990) The design and analysis of spatial data structures. Addison-Wesley, Reading
- Szalay AS, Kunszt PZ, Thakar A, Gray J, Slutz DR, Brunner RJ (2000) Designing and mining multi-terabyte astronomy archives: the Sloan digital sky survey. In: SIGMOD conference, pp 451–462
- Xie D, Li F, Yao B, Li G, Chen Z, Zhou L, Guo M (2016) Simba: spatial in-memory big data analysis. In: SIGSPATIAL/GIS, pp 86:1–86:4
- You S, Zhang J, Gruenwald L (2015) Large-scale spatial join query processing in cloud. In: CloudDB, ICDE workshops, pp 34–41
- Yu J, Wu J, Sarwat M (2015) GeoSpark: a cluster computing framework for processing large-scale spatial data. In: SIGSPATIAL/GIS, pp 70:1–70:4
- Zhang J, Mamoulis N, Papadias D, Tao Y (2004) All-nearest-neighbors queries in spatial databases. In: SSDBM, pp 297–306
- Zhang S, Han J, Liu Z, Wang K, Xu Z (2009) SJMR: parallelizing spatial join with mapreduce on clusters. In: CLUSTER, pp 1–8
- Zhou X, Abel DJ, Truffet D (1997) Data partitioning for parallel spatial join processing. In: SSD, pp 178–196

Queue Mining

Arik Senderovich
Mechanical and Industrial Engineering,
University of Toronto, Toronto, ON, Canada

Definitions

Queue mining is a set of data-driven methods (models and algorithms) for queueing analysis of business processes. Prior to queue mining, process mining techniques overlooked dependencies between cases when answering such operational

questions. To address this gap, queue mining draws from analytical approaches from queueing theory and combines them with classical process mining techniques.

Overview

Modern business processes are supported by information systems that record process-related events in event logs. *Process mining* is a maturing research field that aims at discovering useful information about the business process from these event logs (van der Aalst 2011). Process mining can be viewed as the link that connects process analysis fields (e.g., business process management and operations research) to data analysis fields (e.g., machine learning and data mining) (van der Aalst 2012).

This entry is focused on process mining techniques that aim at answering operational- or performance-type questions such as “does the executed process as observed in the event log correspond to what was planned?,” “how long will it take for a running case to finish?,” and “how should resource capacity or staffing levels change to improve the process with respect to some cost criteria?” (van der Aalst 2011, Chs. 7, 8). Throughout the entry, the term *operational process mining* is used to describe techniques that tackle such questions. Definition of process mining as well as other types of process mining methods, such as social network mining and process model visualization, is beyond the scope of this entry. Below, a historical background is presented. It comprises of existing techniques for operational process mining, along with queue mining methods that extend or improve these methods to accommodate inter-case dependencies.

Historical Background: Emergence of Queue Mining

Historically, process mining techniques overlooked dependencies between cases when conducting operational analysis. For example, state-of-the-art methods for predicting remaining times of running cases considered only historical

data of the case itself (van der Aalst et al. 2011), while the interactions among cases (e.g., through queueing for shared resources) were neglected. The independence assumption is plausible in processes where capacity always exceeds demand. However, in service processes, which are prevalent in numerous application domains (e.g., healthcare, banking, transportation), multiple customer-resource interactions occur, and customers often compete over scarce resources. Consequently, this understanding led to queue mining, which is a set of operational process mining solutions that consider case interactions when answering operational questions. In what follows, the entry surveys operational process mining, specifically, two main streams of work, namely, *model-based process mining* and *supervised process mining*.

Model-Based Process Mining. Early works on performance-oriented process mining were based on the enhancement of control flow models with further data-based information such as decisions, resources, and time (Rozinat et al. 2009). The work of Mărușter and van Beest (2009) extends this approach from modeling and analysis of an existing process to its improvement based on the discovered model. Due to the expressiveness of the discovered models, they can only be analyzed via simulation. Thus, these methods are referred to as simulation mining or model-based process mining, interchangeably.

To discover resource dependencies and roles in the business process, Song and Van der Aalst (2008) propose a method that is based on social network mining. Moreover, Burattin et al. (2013) discover roles and role handover by grouping activities and associating the resulting groups with resources.

In order to create unbiased models, one needs to consider several resource-related factors. For example, it is common to assume that resources work on an activity immediately as it is assigned to them. In reality, however, activities associated with different cases can be processed in batches, when enough work is accumulated. Discovering batch processing of work is key in creating accurate models. Previous work addressed the

discovery of batch processing when mining process models (Nakatumba and van der Aalst 2009; Martin et al. 2017).

Another aspect that can bias process models is related to assumptions on resource speedups and slowdowns when working on activities. In process models, it is often assumed that resources work in constant rates. Yet, previous research has shown that resources often increase and decrease their working rate, depending on stress levels (Wickens et al. 2015). In Nakatumba et al. (2012), the authors investigate the influences of varying processing rates on performance analysis and process discovery. Lastly, acknowledging that resources adhere to a predefined schedule and are not always present to work on activities should be considered when mining event logs for operational purposes (Liu et al. 2012).

The main benefit of model-based techniques is their ability to solve a large set of performance problems. For example, one can use simulation to predict waiting times and propose resource staffing. However, current simulation mining techniques overlook queueing effects by simulating process cases independently of each other and drawing waiting times (and other measures that stem from case dependencies) from fitted distributions. Furthermore, model-based mining techniques tend to overfit reality as it is observed in the event log, and therefore it was shown that simpler performance models often provide accurate results and are more efficient (Senderovich et al. 2015a). The expressiveness problem is partially addressed in Rogge-Solti and Weske (2015), where authors mine non-Markovian stochastic Petri nets (Haas 2002), which are simple formalisms that account for control-flow and exponential durations. In a queue mining technique proposed by Senderovich et al. (2015a), the best of both worlds is combined by a method for simulation mining that captures queueing effects and simplifies the resulting model. This simplification results both in improved efficiency (run-time complexity of the solution) and accuracy.

Supervised Process Mining. Another prevalent stream of work in process mining is the encoding

of event logs into well-established supervised learning methods. Examples for such methods can be found in van der Aalst et al. (2011), de Leoni et al. (2016), Polato et al. (2016), and Leontjeva et al. (2015). In these works, both the behavior and the operational aspects of the process are encoded: the former representing the control-flow perspective and the latter capturing activity durations and resources. While these approaches are often accurate in solving a given mining problem with respect to a set of performance measures observed in the event log (e.g., total time in process), it has two major drawbacks.

First, one cannot use these techniques for quantifying measures that are not directly observable in the log. For example, information that allows one to calculate resource utilization might be missing from the event log, and thus supervised approaches cannot be applied. Second, exploring process improvement directions and sensitivity of process parameters (e.g., durations and arrival rates) is impossible due to the lack of data that describes the effect of changing these parameters. In Gal et al. (2015) and Senderovich et al. (2015b), queue mining solutions bring together the benefits of model-based mining (or simulation mining) and supervised learning. Specifically, these solutions comprise machine learning methods (e.g., decision trees) and queueing models. This combination enables quantifying measures that were not observed in the event log due to the existence of the model. For measures that appear in the event log, one can get accurate solutions by exploiting the strengths of supervised learning (Hastie et al. 2001). Furthermore, one is able to obtain what-if reasoning based on the queueing model, without the need to acquire further data from the changed process. To summarize, queue mining:

- Integrates data-driven queueing models and their analysis into operational process mining (Senderovich et al. 2014, 2015a,b,c, 2016b; Gal et al. 2015).
- Links operational conformance analysis to performance improvement in business

processes (Senderovich et al. 2016b). Specifically, we develop a methodology for the detection of problematic areas in a process (e.g., due to synchronization or resource delays) and propose proper adjustments to queueing policies (selection of the next case to enter service) in these problematic parts.

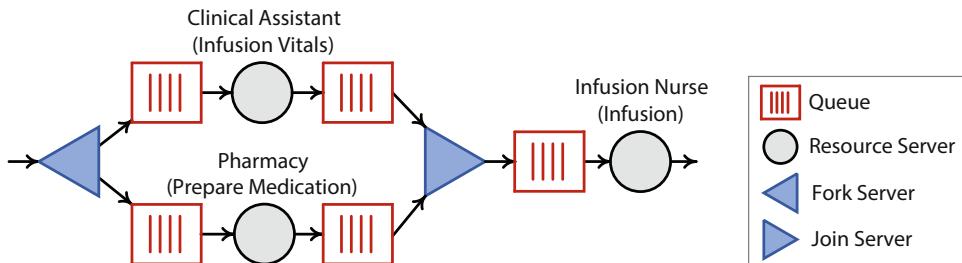
- Offers hybrid methods that combine supervised learning and model-based process mining techniques to improve the accuracy of answering operational questions (Senderovich et al. 2015b; Gal et al. 2015). This combination improves accuracy with respect to applying the two approaches separately.

The proposed queue mining methods were validated based on real-world applications and their corresponding datasets coming from various service domains. The details of these applications and datasets appear in section “[Examples of Applications](#).“ To conclude the overview, the entry offers an elaboration on queue mining basic ideas and methods.

Foundations of Queue Mining

Queue mining methods are based on process models that are referred to as queueing networks. An example of a queueing network appears in Fig. 1. The network depicts the queueing perspective for the patient flow in an outpatient hospital in Boston. The model contains servers (or resources), edges that correspond to routing, three resource queues (preceding resources), and two synchronization queues (succeeding resources). Queues are defined implicitly before and after their respective servers. The example, Fig. 1, contains three resource constructs as well as a fork and a join. The fork and join constructs correspond to AND gateways in control-flow models.

Discovery of a queueing network from an event log involves the identification of the network structure or control-flow (see van der Aalst 2011, Ch. 5 and the references therein), an estimation of routing probabilities (Senderovich et al. 2016b), and a characterization of server



Queue Mining, Fig. 1 Patient flow: an outpatient hospital in Boston

dynamics (Senderovich et al. 2015b). When a queueing network is discovered, one may employ numerous techniques from queueing theory to analyze performance based on the network (see Bolch et al. 2006 and the references therein). For example, one may use a queue-length-type predictor to estimate the expected waiting time in a station (Ibrahim and Whitt 2009; Senderovich et al. 2015b). These queue mining techniques are referred to as *model-based queue mining*.

second approach for the analysis is by using the resulting queueing network in combination with historical data (i.e., the event log) for constructing an improved supervised learning framework (Gal et al. 2015). The latter approaches are referred to as *supervised queue mining*. The two approaches clearly correspond to the ideas of *model-based process mining* and *supervised process mining*, respectively. Below, the entry briefly explains the two approaches based on a prediction operational problem.

Model-Based Queue Mining. Depending on the complexity of the resulting queueing network, we may apply different queueing analysis techniques ranging from simulation to tractable steady-state analysis (Bolch et al. 2006). If the model is a complex network of queues with concurrency and multiple customer classes, the analysis will rely mainly on stochastic simulation (Rubinstein 1986; Glynn and Iglehart 1988) or approximation techniques (Whitt 1983; Chen and Yao 2013). In contrast, for simple networks that do not present time fluctuations, one may apply closed-form results from, at

steady state, classical queueing theory (Kelly 1975; Bramson 2008; Jackson 1957).

Here we give an example of a discovered single-station single-class network with $G/M/s$ server dynamics, i.e., a general arrival process with i.id. inter-arrival times, i.id. exponentially distributed service times, and s independent homogeneous service providers. Further, the service policy is assumed first-come first-served; we propose methodologies for learning the policy from data and testing whether an assumed policy holds (Senderovich et al. 2014, 2015c).

Based on the resulting model, we may apply a well-known predictor named the queue-length predictor (QLP) (Whitt 1999). As its name suggests, it exploits the queue-length ahead of the customers whose delay time we wish to predict. The QLP was shown to work well with no abandonments. However, it fails when abandonments are prevalent (e.g., in call center scenarios) (Senderovich et al. 2015b). Further, queue-length predictors as well as other queueing theory techniques are mined, tested, and extended based on real-world call center data in Senderovich et al. (2015c).

Supervised Queue Mining. Supervised queue mining considers incorporation of queueing analysis into machine learning methods. Currently, the idea comprises two basic approaches: (1) feature enrichment and (2) model adaptation.

Approach 1: Feature Enrichment. In this approach, we define (feature) construction functions that transform event data into queueing features. These functions rely on the discovered

queueing model, which contains information that reflects dependencies among running cases. The resulting queueing features are assumed to better explain operational performance measures. Note that with this approach, the learning method (e.g., a linear regression) is used out-of-the-box.

To illustrate the approach, consider an event log with the following basic features: case identifier, activity, and timestamp. Applying a construction function that is based on the discovered queueing network to these features can result in new queueing features, e.g., waiting time of the head-of-the-line and current queue-length. Subsequently, these queueing features are plugged into a learning method of choice (e.g., linear regression, regression trees, and random forests). This substitution was found to significantly improve prediction accuracy in Senderovich et al. (2015b) and Gal et al. (2015).

Approach 2: Model Adaptation. An alternative approach to the aforementioned feature enrichment is an adaptation of the learning method itself, based on the discovered queueing network. For example, in Gal et al. (2015), a gradient tree boosting method that comprises multiple learners (decision trees) was extended (Friedman 2001). Every learner is built on top of the other iteratively. In Gal et al. (2015), the first predictor was based on a well-established result in queueing theory, which we refer to as the snapshot principle (Gal et al. 2015; Senderovich et al. 2015b). The other predictors were plain decision trees, which were constructed via boosting based on the initial queueing predictor. In Gal et al. (2015), it was shown empirically that the proposed method improves prediction accuracy both over the gradient tree boosting algorithm without the snapshot predictor and over the snapshot predictor when it is applied without a learning procedure.

As a concluding remark, it is worth mentioning that queue mining is limited neither to the discovery of a single queueing model nor to the application of a single technique for solving one operational mining problem. On the contrary, given a set of problems, one is advised to discover different queueing models for the different types of analysis. Our studies show

that, for time prediction, one would prefer a simple queueing model over complex simulation models. The former gives rise to accurate and efficient solutions. In contrast, for conformance checking, a complex queueing network is preferable, since it allows one to capture deviations in multiple dimensions, e.g., durations, resources, and customer classes. Therefore, queue mining can be viewed as a variety of solutions that can be combined to solve a set of operational mining problems.

Examples of Applications

In this part, the effectiveness of queue mining techniques is demonstrated on three real-world applications, namely, online delay prediction, root cause analysis, and process improvement. Furthermore, experiments that show the empirical value of queue mining were based on real-world data from three domains: banking (a bank's call center), transportation (city buses), and healthcare (an outpatient hospital).

Online Delay Prediction

The first problem that was addressed by queue mining was to predict delays of arriving cases, in real time. To this end, two sets of predictors were considered: learning-based and model-based.

The first set of predictors took traditional approaches to supervised learning as a starting point and extended the existing regression-based techniques for time prediction, e.g., (van der Aalst et al. 2011), to consider queues and system load. A second set of predictors originated from queueing theory (Hall 1991; Bolch et al. 2006) and leveraged properties of a queueing model, potentially under a widely known congestion law (the snapshot principle).

Further, queue mining studies have found that delay prediction is sensitive to classifications of cases. In a single-class setting, all cases are of a uniform type, and delay prediction relates to a single queue of cases. In a multi-class setting, in turn, cases are classified by a certain type that

influences how cases are enqueued. Hence, delay prediction relates to a specific case type.

By taking the above considerations into account, it was shown that queue mining techniques significantly improve delay prediction (Senderovich et al. 2015b). The experiments were conducted on several real-world datasets, and the developed methods yielded an improvement in various performance measures.

The results were extended to networks of queues with predefined and random routing in Gal et al. (2015) and Senderovich et al. (2016a), respectively.

Root Cause Analysis

The next application of queue mining was root cause analysis (RCA) (Senderovich et al. 2016b). The goal was to discover deviations between actual executions of scheduled business processes and the underlying schedule or plan that govern such processes. Example for such processes can be found in healthcare (appointment book is the schedule) and in public transportation (bus timetable is the schedule).

The proposed method comprises of two main steps. First, deviations in performance measures are quantified. For example, if waiting times were longer than the corresponding planned buffers, they are classified as a deviation. Clearly, these deviations must be tested statistically for their significance (Senderovich et al. 2016b).

Next, these deviations are connected to local deviations in the queueing network that arises from the data. The network's building blocks (arrival rates, service times, capacities, and service policies) are compared against the predefined counterparts from the schedule. For example, if 5 doctors were scheduled between 8:00AM and 9:00AM, and the number of doctors during that time has a distribution, the actual number that arrived to work (say 6) is compared to the median via a statistical test. If a deviation is found in these building blocks, then a root cause for waiting time discrepancies is announced.

The underlying assumption in this method is that deviations from schedule drive local differ-

ences in terms of performance measures (e.g., waiting times and queue lengths).

An example for a root cause analysis that was conducted on a real-world hospital's dataset is as follows. In a large cancer hospital in the United States, queue mining techniques were able to show that deviations in production policy of chemotherapeutic drugs in the pharmacy were a root cause for the delays in the process.

Process Improvement

The aforementioned root cause analysis detects parts of the process that fail to conform (conceptually or operationally) to a given schedule. Here, the focus switches is on how to handle the detected problems by introducing a methodology for process improvement, which combines data-driven analysis via the queueing network, and principles from scheduling research (Pinedo 2012).

Provided with the queueing network (see, e.g., Fig. 1) which corresponds to some underlying business process, one may aim at a local improvement of the service policy, given problematic areas. We assume that splits and joins have a single layer of resource nodes, a plausible assumption since multiple stages can be aggregated into such a construct (Senderovich et al. 2015a).

default, processes often operate under the earliest due date (EDD) first service policy per node, thus “optimizing” schedule-related performance measures (e.g., non-punctuality). Assuming that all cases are available at the beginning of the scheduling horizon, it is indeed optimal to use the EDD policy. However, when cases arrive into the system at different times (according to schedule), one can show that the EDD policy can be improved to achieve lower tardiness. Moreover, using queue mining, one can show that without losing punctuality, the proposed algorithms may also improve other performance measures such as flow time. This is achieved by considering synchronization delays in concurrent processing and by partially reordering EDD-ordered cases in a first-come first-served order.

Future Research Directions

Future research in queue mining can be divided into two categories with respect to the following dimensions: (1) model-based vs. supervised queue mining and (2) predictive vs. prescriptive analysis.

The first dimension corresponds to the representational choices of the underlying data and process. Model-based approaches can be extended with further components of queueing models. For example, the queueing networks can be enhanced with additional constructs, such as shared resources, which were not previously considered as part of queue mining. For supervised queue mining, future work may include a feature encoding method that would not assume any prior process knowledge. Specifically, queueing insights provide with natural metrics to measure distances between running cases. This allows for the abandonment of strict assumptions on the underlying queueing system. For example, the snapshot features can be considered special cases of more general temporal distance functions that measure proximity between sequences.

The second dimension addresses the questions that one wishes to answer with queue mining. Continuing the predictive direction, one could consider additional tasks beyond time prediction. For example, one may analyze the utilization profile of resources and aim at predicting the time that resources will return from breaks and other types of idle periods. For the prescriptive direction, a natural extension of queue mining would be to further advance into optimization of processes. For instance, having observed historical decisions (e.g., the ordering of customers waiting in queues) can an optimal decision-making policy be automatically devised from data. Although the task bears similarities to predictive analysis, it also differs in that the decision policy must also accommodate situations that were not observed in the data. Hence, a combination of statistical causal inference methods, and queueing modeling, must be considered. Consequently, queue mining would serve the bridge between statistical analysis, optimization, and process management

and mining. Lastly, one of the current topics considered in future work is the construction of a unified tool that brings all aspects of queue mining (modeling and analysis) in a software package. This would enable researchers and practitioners to add and use implementations of novel queue mining methods.

Cross-References

- [Business Process Performance Measurement](#)
- [Data-Driven Process Simulation](#)
- [Decision Discovery in Business Processes](#)
- [Multidimensional Process Analytics](#)

References

- Bolch G, Greiner S, de Meer H, Trivedi KS (2006) Queueing networks and Markov chains – modeling and performance evaluation with computer science applications, 2nd edn. Wiley, Hoboken
- Bramson M (2008) Stability of queueing networks. Springer, Berlin/Heidelberg
- Burattin A, Sperduti A, Veluscek M (2013) Business models enhancement through discovery of roles. In: 2013 IEEE symposium on computational intelligence and data mining (CIDM). IEEE, pp 103–110
- Chen H, Yao DD (2013) Fundamentals of queueing networks: performance, asymptotics, and optimization, vol 46. Springer Science & Business Media, New York
- de Leoni M, van der Aalst WMP, Dees M (2016) A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf Syst 56:235–257
- Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29(5):1189–1232. <https://doi.org/10.1214/aos/1013203451>
- Gal A, Mandelbaum A, Schnitzler F, Senderovich A, Weidlich M (2015) Traveling time prediction in scheduled transportation with journey segments. Inf Syst 64:266–280
- Glynn PW, Iglehart DL (1988) Simulation methods for queues: an overview. Queueing Syst 3(3):221–255
- Haas PJ (2002) Stochastic petri nets: modelling, stability, simulation. Springer, New York
- Hall RW (1991) Queueing methods: for services and manufacturing. Prentice Hall, Englewood Cliffs
- Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer series in statistics. Springer, New York
- Ibrahim R, Whitt W (2009) Real-time delay estimation based on delay history. Manuf Serv Oper Manag 11(3):397–415
- Jackson JR (1957) Networks of waiting lines. Oper Res 5(4):518–521

- Kelly FP (1975) Networks of queues with customers of different types. *J Appl Probab* 12(3):542–554
- Leontjeva A, Conforti R, Francescomarino CD, Dumas M, Maggi FM (2015) Complex symbolic sequence encodings for predictive monitoring of business processes. In: Proceedings of business process management – 13th international conference, BPM 2015, Innsbruck, 31 Aug–3 Sep 2015, pp 297–313
- Liu T, Cheng Y, Ni Z (2012) Mining event logs to support workflow resource allocation. *Knowl-Based Syst* 35:320–331
- Martin N, Swennen M, Depaire B, Jans M, Caris A, Vanhoof K (2017) Retrieving batch organisation of work insights from event logs. *Decis Support Syst* 100: 119–128. <https://doi.org/10.1016/j.dss.2017.02.012>
- Märuster L, van Beest NR (2009) Redesigning business processes: a methodology based on simulation and process mining techniques. *Knowl Inf Syst* 21(3): 267–297
- Nakatumba J, van der Aalst WMP (2009) Analyzing resource behavior using process mining. In: Rinderle-Ma S, Sadiq SW, Leymann F (eds) Business process management workshops, BPM 2009 international workshops, Ulm, 7 Sept 2009, Revised Papers. Lecture notes in business information processing, vol 43. Springer, pp 69–80. https://doi.org/10.1007/978-3-642-12186-9_8
- Nakatumba J, Westergaard M, van der Aalst WMP (2012) Generating event logs with workload-dependent speeds from simulation models. In: Proceedings of advanced information systems engineering workshops – CAiSE 2012 international workshops, Gdańsk, 25–26 June 2012, pp 383–397. https://doi.org/10.1007/978-3-642-31069-0_31
- Pinedo M (2012) Scheduling. Springer, New York
- Polato M, Sperduti A, Burattin A, de Leoni M (2016) Time and activity sequence prediction of business process instances. *CoRR* abs/1602.07566
- Rogge-Solti A, Weske M (2015) Prediction of business process durations using non-markovian stochastic petri nets. *Inf Syst* 54:1–14
- Rozinat A, Mans R, Song M, van der Aalst W (2009) Discovering simulation models. *Inf Syst* 34(3):305–327
- Rubinstein RY (1986) Monte Carlo optimization, simulation, and sensitivity of queueing networks. Wiley, New York
- Senderovich A, Weidlich M, Gal A, Mandelbaum A (2014) Mining resource scheduling protocols. In: Business process management. Springer, pp 200–216
- Senderovich A, Rogge-Solti A, Gal A, Mendling J, Mandelbaum A, Kadish S, Bunnell CA (2015a) Data-driven performance analysis of scheduled processes. In: Business process management. Springer, pp 35–52
- Senderovich A, Weidlich M, Gal A, Mandelbaum A (2015b) Queue mining for delay prediction in multi-class service processes. *Inf Syst* 53:278–295
- Senderovich A, Weidlich M, Gal A, Mandelbaum A, Kadish S, Bunnell CA (2015c) Discovery and validation of queueing networks in scheduled processes. In: Advanced information systems engineering. Springer, pp 417–433
- Senderovich A, Shleyfman A, Weidlich M, Gal A, Mandelbaum A (2016a) P ^3-fold: optimal model simplification for improving accuracy in process performance prediction. In: Proceedings of business process management – 14th international conference, BPM 2016, Rio de Janeiro, 18–22 Sept 2016, pp 418–436
- Senderovich A, Weidlich M, Yedidsion L, Gal A, Mandelbaum A, Kadish S, Bunnell CA (2016b) Conformance checking and performance improvement in scheduled processes: a queueing-network perspective. *Inf Syst* 62:185–206
- Song M, Van der Aalst WM (2008) Towards comprehensive support for organizational mining. *Decis Support Syst* 46(1):300–317
- van der Aalst WMP (2011) Process mining – discovery, conformance and enhancement of business processes. Springer, Berlin/Heidelberg
- van der Aalst WMP (2012) Process mining: overview and opportunities. *ACM Trans Manag Inf Syst* 3(2):7
- van der Aalst WMP, Schonenberg MH, Song M (2011) Time prediction based on process mining. *Inf Syst* 36(2):450–475
- Whitt W (1983) The queueing network analyzer. *Bell Syst Techn J* 62(9):2779–2815
- Whitt W (1999) Predicting queueing delays. *Manag Sci* 45(6):870–888
- Wickens CD, Hollands JG, Banbury S, Parasuraman R (2015) Engineering psychology & human performance. Psychology Press, New York

R

R Language: A Powerful Tool for Taming Big Data

Norman Matloff¹, Clark Fitzgerald², and Robin Yancey³

¹Department of Computer Science, University of California, Davis, CA, USA

²Department of Statistics, University of California, Davis, CA, USA

³Department of Electrical and Computer Engineering, University of California, Davis, CA, USA

It should be noted that Big Data can be “big” in one of two ways, phrased in terms of the classical $n \times p$ matrix representing a dataset:

- **Big-n:** Large number of data points.
- **Big-p:** Large number of variables/features.

Both senses will come into play later. For now, though, back to R. Some general information about the language will be presented first, as foundation for the Big Data aspects.

Overview

Definitions

The R language (R Core Team 2017; Chambers 2008; Matloff 2011) is currently the most popular tool in the general data science field. It features outstanding graphics capabilities and a rich set of more than 10,000 library packages to draw upon. (Other notable languages in data science are Python and Julia. Python is popular among those trained in computer science. Julia, a new language, has as top priority producing fast code.) Its interfaces to SQL databases and the C/C++ language are first rate. All of this, along with recent developments regarding memory issues, makes R well poised as a highly effective tool in Big Data applications. In this chapter, the use of R in Big Data settings will be presented.

In terms of syntax, R, along with Python, C/C++, and many others, is ultimately a descendant of ALGOL, and thus a programmer in one of those languages can quickly pick up at least a rough “reading knowledge” of R.

As with Python, R is an *interpreted* language, meaning it is not translated to machine code, unlike the C/C++ language. The interpreted nature of R brings up possible performance issues, a topic to be discussed in section “[Uniprocessor Performance Issues](#)”.

From a programming style point of view, R (to various degrees) follows the *object-oriented* and *functional programming* philosophies. Some computer scientists believe that they lead to clearer, safer code, and this has influenced the design of R. This in turn will have implications for all R users, as will be explained.

Vectors and matrices are similar to one- and two-dimensional arrays in C. A *list* is like a vector, but with possibly different modes. A *data frame* looks like a matrix, but its columns can be of different modes, e.g., numeric in some, character in others, and logical in still others. These structures are intrinsic parts of the R language, as opposed to add-ons in the case of Python.

Subsetting is a major operation in R. For instance,

```
> d <- data.frame(x=c(5,12,13), y=c(8,88,888))
> da <- d[c(1,3),]
> d
      x     y
1 5     8
2 12    88
3 13   888
> da
      x     y
1 5     8
3 13   888
> row.names(da)
[1] "1" "3"
```

Such information could be quite useful and is attained without adding an extra column in the data frame that may need to be excluded in subsequent statistical computations.

One of R's most lauded features is its ability to produce beautiful, highly expressive graphics, in a manner accessible to even nonspecialists. Base R graphics is used for simpler plots or for advanced applications operating at a more finely detailed level (Murrell 2011; Chang 2013). For higher-level applications, the **ggplot2** (Wickham 2016) and **lattice** (Sarkar 2008) packages are quite powerful and are widely used. Another notable graphics package is Plotly (Plotly Technologies Inc. 2015), which produces especially esthetically appealing figures. A quite usable R interface is available.

`m[c(1,4,5),]`

is the submatrix of the matrix **m** consisting of rows 1, 4, and 5 of that matrix.

A seldom cited but widely used and very handy feature of R is the ability to set names of various data elements. For an example of how this can be useful, suppose one has a large data frame, and then split it into chunks of rows according to some criterion. The original row names will be retained:

Uniprocessor Performance Issues

As mentioned, R is an interpreted language, which raises performance concerns. The chief remedy is *vectorization*, referring to the fact that vector operations should be used instead of loops.

Vectorization and R as a Functional Language

First, the functional language nature of R is reflected in the fact that, for instance, the `+` operator is actually a function. The expression `x+y`, for vectors `x` and `y`, is actually the function call `'+'(x,y)`. The key point is that function is written in C, not in R. (This should not be confused with the fact that R itself – meaning the R interpreter – is written in C.) Thus C-level speed is attained, which could be an issue for very long vectors.

By the way, `<-` is a function too. So, `z <- x + y` is actually

```
'<-'(z, '+'(x, y))
```

It turns out that many R expressions can be vectorized. For instance, the `ifelse()` function vectorizes the classical if–then–else construct:

```
> x <- 1:3
> y <- c(5, 12, 13)
> ifelse(y > 8, x, x+1)
[1] 2 2 3
```

Interfacing R to C/C++

But in some R applications, vectorization is not enough; the general speed of C must be obtained directly, i.e., entire R functions must be written in C. For instance, `dplyr`, a popular R package for manipulation of data frames, is written partly in C, and `data.table`, an extremely fast (but compatible) alternative to data frames (section “[Data Input/Output, etc. with data.table](#)”), is based largely on C++.

This is a standard approach in the use of scripting languages such as R and Python. Typically only a portion of one’s code requires high performance. It thus makes sense to write that portion in C/C++ while retaining the convenience and expressiveness of R/Python for most of one’s code.

R has two main functions, `.C()` and `.Call()`, for calling C/C++ functions from R code. They are fairly easy to use, and many R programmers use another popular R package, `Rcpp`, that aims to further simplify the R/C interface process (Eddelbuettel 2013).

Interfaces that allow R code and Python to call each other are also available, such as the `rpy2` package.

R and Big Data

The sheer size of Big Data calls for parallel computation, either on multicore machines or clusters. (Define a *physical* cluster to be a set of independent machines connected via a network. Later, *virtual* clusters will be introduced, which

will consist of a number of R invocations running independently. They may be running on a physical cluster, a multicore machine, or a combination of the two.)

Note carefully that the issue is not only one of computation time but also of memory capacity. The latter point is just as important as the former; an application may be too large for a single machine, but if the data is broken into chunks and distributed to the nodes in a cluster, the application might be accommodated.

A detailed treatment of parallel computation in R is given in Matloff (2015). An overview is presented here.

Memory Issues

Earlier versions of R, 2.x.x, set a limit of $2^{31} - 1$ bytes for object size. For modern 64-bit machines, this limited vectors, for instance, to about 250 million numbers, which was not sufficient in certain Big Data contexts. R 3.x.x changed this limit to about 2^{52} , more than enough for even the biggest of Big.

Thus the constraint faced by most people in the R community is not in R limitation on object size, but in the memory sizes of their machines. R stores all objects in memory, so memory size may be an issue in R applications in Big Data.

The bigmemory Package

One solution is the `bigmemory` package (Kane et al. 2013), which offers programmers a choice between storing objects in memory or on disk, with the latter option being attractive on machines with only moderate amounts of RAM. The key point is that from the programmer’s point of view, the data look like they are in memory.

Objects of class `bigmemory` must be of matrix type. A vector is represented as a one-row or one-column matrix and a scalar as a 1×1 matrix.

Recall that operators like `+` are actually implemented as functions. The array indexing operator `[` is another example of this. What `bigmemory` does is replace R’s `[` function by special-purpose C++ code that relays the requested data reference to either memory or disk, according to what the user’s code originally requested.

Another advantage of **bigmemory** among many of its users is that it provides a workaround to R’s “no side effects” policy, which comes from R’s status as a functional language. What this means is that a function call should not change any of its arguments.

For example, the statement

```
sort(x)
```

will *not* change **x**. It returns the sorted version of **x**, but **x** itself doesn’t change. If one wants that change, one needs to write

```
x <- sort(x)
```

Now consider the innocuous-looking statement

```
z[5] <- 8
```

As noted, assignment is a function call, in this to the function '`[<-`'. Under a no-side-effects policy, this would be implemented internally as

```
z <- '[<-'(z, 5, 8)
```

with the right-hand side returning a *new* copy of **z**. In a Big Data context, the creation of this new vector could be quite costly in execution time.

Recent versions of R try to avoid this to some extent, but by placing **z** in a **bigmemory** object, the problem is definitely avoided.

Other Ways to Circumvent Memory Size Problems

As noted, one solution to this problem is to exploit the fact that **bigmemory** can store an object on disk but have it appear to the programmer as if it were in memory. Two other methods are common:

- If one is running on a cluster, one can use the data in distributed form, with one chunk of the data at each cluster node. The chunks may fit into memory even if the full data does not.
- If the data is stored on disk in an SQL database, various R packages for interfacing to SQL are available, such as **RMySQL**.

In such settings, it may also be useful to then use a Software Alchemy approach; see sections “[Distributed Computation](#)” and “[Software](#)

[Alchemy](#)”. Or in the cases in which the application algorithm needs only sums, such as linear regression models, one can read in data chunk-by-chunk, updating the sums at each stage.

Threaded Code

The standard mechanism used for parallel computation by multicore programs in C/C++ is *threaded* code (Matloff 2015; Breshears 2009). Here several copies of one’s code run simultaneously, sharing global data. That latter trait is key. For many algorithms, efficient parallelization requires shared-memory computation. If one has a multicore machine (or a manycore coprocessor, such as the Intel Xeon Phi) and the problem can fit into available memory, threaded programming of some kind is needed.

Roughly, here is how threads work. Say, for instance, the threads are executing some iterative algorithm. At the end of each iteration, each thread will need to know the results of the executions of the various threads, as this will determine the course of action in the next iteration. Accessing shared variables for this kind of action is the essence of threaded programming.

Threads systems must have some form of “lock” variables to avoid *race conditions*. That term refers to a situation in which two threads attempt to change a certain variable at approximately the same time, possibly resulting in incorrect results. A lock variable assures that only one thread at a time can access the sensitive variable.

Threads in C/C++

Say one wishes to find the sum of a very long array **x** of length **n**, with four threads. Each thread would have an ID, 0, 1, 2, or 3, stored in the variable **myID**. Each thread would have its own running sum, stored in **mySum**. The code would look something like this:

```
parfor (i = 0; i < n; i++) {
  if (i % 4 == ID) mySum +
    = x[i];
}
// lock the lock (not shown)
tot += mySum;
// unlock the lock (not shown)
```

Here “parfor” means that all the threads would execute the loop simultaneously. (The variables **myID** and **mySum** would be local to each thread, while **tot** would be global to all.)

Typically one does not write threaded code directly, opting instead to use a higher-level interface to threads, the most widely used being OpenMP (Breshears 2009). Intel’s Threads Building Blocks (Reinders 2007) may produce faster code, but requires more programming skill (and patience), as it is more complex and uses C++ templates.

Threads in R

Scripting languages typically do not offer true threads programming. Python threads, for instance, must use something called the global interpreter lock, which prevents parallel execution of threads. Julia threads are, as of this writing, experimental.

R does not offer threading at all. However, one can still write threaded applications, in one of two ways:

- If one wants to limit one’s code to R, a quasi-threading environment is provided by the **Rdsm** package (Matloff 2015). It runs on top of the **bigrmemory** system discussed earlier, as well as atop the **parallel** package that is included with base R (section “[partools](#)”).

To see how this works, let’s again consider our earlier example. **Rdsm**, in creating a shared variable **z** – in memory, not on disk – will save on disk information as to where in memory **z** resides. If one then has multiple invocations of R running simultaneously, they can all access that same memory location, and thus share **z**, thus attaining the essence of threaded code.

In the array-summing example above, the programmer would call a **parallel** function, **splitIndices()**, to explicitly assign values of **i** to the various threads. Instead of an explicit loop, the code would be vectorized, looking something like this, using the built-in R function **sum()**:

```
myIndices <- splitIndices
(etc .)[[myID]]
```

```
mySum <- sum(x[1, myIndices])
# lock the lock variable
# (not shown)
totx[1,1] <- totx[1,1] + mySum
# unlock the lock variable
# (not shown)
```

Here **x** and **totx** are variables in shared memory.

- One can have one’s R code access C/C++ code that does threading, say using OpenMP. In the above array-summing example, the C code could be used without change. (The R system file **R.h** must be included.)

The **data.table** package is written largely in C++ and makes use of OpenMP.

Another important example is the Basic Linear Algebra Subroutines (BLAS) library. Lots of BLAS libraries exist, and when building R one has the option of using one other than what is included in the R source code. One that is very fast is OpenBLAS, which again uses OpenMP to run threads.

Message-Passing Parallel Code in R

In addition to the shared-memory world view seen above, another paradigm of parallel computation is *message-passing*. Here, instead of having different processes communicate via shared variables, they explicitly send information to each other.

This approach is generally used on clusters rather than multicore machines, though many users do use it on the latter as well. In either case, it must be kept in mind that a major source of speed-sapping overhead is the time spent sending the messages, especially in the cluster case. For this reason, these approaches are generally efficient only in settings in which the time between network accesses is long. This in turn means the application is such that a large amount of computation is done between network accesses, a situation known as *coarse-grained parallelism*.

It should be repeated, though, that for many large problems, memory size is the overriding constraint. In such cases, a message-passing approach on a cluster may be the only feasible solution.

The “parallel” Package

A workhorse of parallel computation in R is the built-in **parallel** package. Its main virtue is its conceptual simplicity; users can readily write their own **parallel** code after seeing a few examples.

The package was adapted from the previous user-contributed packages **snow** and **multicore**. The focus here will be on the former case, which implements *scatter/gather* operations.

As with **Rdsm** (and **ddr** below), there will be many independent invocations of R. When the

user starts R, let’s refer to that as the “manager” invocation. It launches new invocations of R, termed “workers.” One writes code to run on the manager that distributes data to the workers – the *scatter* phase – and sends the workers commands to execute on the data given them. They return results, which are *gathered* at the manager. The latter pieces them together to obtain the desired outcome.

Here is a simple example, again involving an array-summing operation, in parallel over the workers:

```
library(parallel)
cls <- makeCluster(2)
x <- c(5,12,13,8,88)
clusterExport(cls, 'x')
clusterApply(cls, 1:2, function(i) myID <- i)
clusterEvalQ(cls, library(parallel))
clusterEvalQ(cls, myIndices <-
  splitIndices(5,2)[[myID]])
Reduce(sum(clusterEvalQ(cls, sum(x[myIndices])))
```

All this runs on the manager. It first makes a virtual cluster of two workers, meaning two new invocations of R connected to the manager. It then creates an example vector **x** and “exports” it to the workers. (This is not very efficient, as each worker needs only part of **x**.) Next, it sets up an ID at each worker, by applying the given function to (1,2) at the workers. Then it loads the **parallel** package at each worker; the function **clusterEvalQ()** simply instructs each worker to execute the given code locally. Next it runs

```
clusterEvalQ(cls, myIndices
  <- splitIndices(5,2)[[myID]])
```

This instructs each worker to assign to **myIndices** as indicated. The call to **splitIndices()** returns

```
[[1]]
[1] 1 2

[[2]]
[1] 3 4 5
```

i.e., an R list whose first component is (1,2) and the other (3,4,5). The purpose of this is to have Worker 1 handle elements 1 and 2 of **x**, while Worker 2 will handle the rest. After execution of the above statement, the variable **myIndices** will have the value (1,2) at Worker 1 and (3,4,5) at Worker 2.

Finally, there is the **Reduce()** call. Inside is a call to **clusterEvalQ()**, which has each worker sum its portion of **x**. That call returns to the manager an R list consisting of the two sums, 17 and 109. **Reduce()** then repeatedly applies **sum()** to each element of that list, resulting in the full sum 126.

As can be seen, compared to the shared-memory paradigm, the programmer must do quite a bit of work just for this simple operation. This is typical of message-passing systems. Many programmers use the **foreach** package (Weston 2017) as a simpler, more convenient wrapper for **parallel** in situations in which the main goal is to parallelize a loop.

Rmpi

A major limitation of **parallel** is that communication is possible only between a worker and the manager. The **Rmpi** package (Yu 2014), based on the famous C/C++/FORTRAN message-passing library MPI (Nielsen 2016), is much more general, allowing direct communication from any worker to any other. For some applications, this can increase performance tremendously.

Rmpi is more complex to program (and to configure) and thus is too involved to present in further detail here.

Distributed Computation

One sees much in the press about the Hadoop and Spark frameworks. These have been proven quite effective on very large systems for simple tabulatory computations such as sums, counts, and data grouping. And they do have R interfaces for them, e.g., **sparklyr** (Luraschi et al. 2017).

However, a major drawback to these frameworks is that one essentially must do a global sort after each operation, whether needed or not, potentially quite a drain on speed. Though Spark is a major improvement over Hadoop, neither is well-suited to the more complex statistical and data-wrangling operations typical in data science. Here two alternatives are presented.

Both of the packages in this section operate with distributed data. For instance, say **z** is a data frame with 10 million rows, and one has 4 workers in a cluster in the above sense. One could store 2.5 million rows at each worker, and to the degree possible and as long as possible, the data is

```
clusterEvalQ(cls, convert factors to dummy variables)
clusterEvalQ(cls, replace NA values by means)
clusterEvalQ(cls, remove outliers, say by a "3 sigma" rule)
calm(cls, regression formula)
```

That last function call implements Software Alchemy, to be explained in section “[Software Alchemy](#)”. It runs the R **lm()** linear model function at each cluster node and combines the results to obtain the overall estimated regression coefficients.

kept distributed in that manner, over the course of many data and statistical operations. In the above scatter/gather terms, one scatters but then avoids gathering for as long as possible. The goal, of course, is to avoid costly network communication delays. Informally let’s call this policy “Leave It There” (LIT).

LIT is a very simple idea, yet a very powerful one.

partools

Both Hadoop and Spark are typically run on distributed file systems. The **partools** package (Matloff et al. 2017a), which runs on top of **parallel**, is predicated on the view that this is the best approach, but that the operation structures of Hadoop and Spark, with frequent network communication and disk read/writes, are unsuited for statistical/data science applications.

Again, consider a simple example of four workers. One might store the data in four files, **x.1**, **x.2**, and so on. The manager code would execute something like

```
file.read(c1s, 'x', 'x', 1)
```

Here **cls** is the name of the virtual cluster; the basename of the distributed file is “**x**”; and one wishes the distributed data frame at the virtual cluster nodes to also be named “**x**.”

Execution of the distributed program proceeds mainly with manager calls to **clusterEvalQ()**, instructing the workers to perform certain tasks. For instance, in a linear regression problem, the code at the manager might look like this:

R

Note that the data, even after various operations have been performed, is *still* distributed, available for further distributed operations, again following the LIT philosophy. And at the end of the session, the user can save the modified distributed data frame to a distributed file.

Numerous utility functions are available, including ones to convert data between distributed and monolithic forms: **distribsplit()**, to convert a monolithic data frame to a distributed one, and **distribcat()**, to go in the opposite direction.

The package also includes a number of statistical and tabulatory functions to do non-LIT operations, such as **distribagg()**, which performs the R **aggregate()** function at each node and then combines appropriately.

In addition, **partools** offers direct point-to-point communication between cluster nodes, which greatly extends the ability to write fast parallel code. One of the applications of that facility is an implementation of the Hyperquicksort distributed sorting algorithm.

```
library(ddR)
chunk <- 2
nc <- 5
n <- chunk * nc
x <- as.darray(matrix(rnorm(n)), psize = c(nc, 1))
y <- as.darray(matrix(rnorm(n)), psize = c(nc, 1))
setMethod("-", signature(e1 = "ParallelObj",
                        e2 = "ParallelObj"),
           function(e1, e2) {
             dmaply(`-`, e1, e2, output.type = "darray",
                   combine = "cbind")
           })
delta <- x - y
as.vector(collect(delta))
```

The package uses R’s S4 object-oriented programming model to define operations on distributed objects, in this case subtraction. The S4 function **setMethod()** is shown above to define subtraction between two objects of class **ParallelObj**, creating a third object of that class.

Software Alchemy

Real-world Big Data applications on parallel platforms tend not to be “embarrassingly parallel,” due to inter-process communication costs. This can severely limit potential speedup.

ddR

In designing a package for distributed computation, one desirable trait would be (at least near) compatibility with existing serial code, enabling software reuse. The **ddR** package (distributed data in R) satisfies this criterion, defining distributed versions of some R data structures, including arrays, lists, and data frames. Several algorithms have been implemented in this framework, such as generalized linear models and random forests. The package runs on top of **parallel** or another HP product, **distributedR**.

Here is an example of **ddR** code to initialize and subtract two distributed matrices.

Fortunately, for statistical/machine learning applications, this frequently can be resolved by a method also known as “Software Alchemy” (SA) in Matloff (2016). Here the data is divided up into chunks which are distributed to each available cluster node, and one averages all the results of the same computation from each process. In this way we “alchemically” transform the original problem into an embarrassingly parallel problem that produces a statistically equivalent result: The asymptotic covariance matrix of the SA estimator can be shown to be the same as that of the estimator based on the full set, i.e., it achieves

the same accuracy. (It should be noted that the speed of convergence of the asymptotics may depend on p , the number of variables/features in the full dataset (Bühlmann et al. 2016).)

Theoretically, the speedup can be derived in time complexity terms. Suppose the complexity of the full algorithm is known to be $O(n^d)$, where n is the number of data points. Let r denote the number of worker threads. Then the SA complexity will be approximately $O[(n/r)^d] = O(n^d/r^d)$, a speedup of about r^d . If $d > 1$, this would be a superlinear speedup, a rarely seen event in the parallel computation world, but that has appeared experimentally in plots of the output using SA (Matloff 2016). (The same analysis shows, interestingly, that one obtains a speedup even in the serial case, for $d > 1$.)

Our experiments so far have shown SA to be useful for several widely used machine learning algorithms. For example, consider the famous Forest Cover dataset in the UCI Machine Learning Data Repository (Lichman 2017), consisting of half a million samples for classification into one of seven classes. Using the random forests algorithm, we can produce a speedup of 4.2 without loss in accuracy. This reduces computation time by over 10 min by just taking advantage of a standard four-core machine with six processes. (The machine used here has *hyperthreading*, which makes it perform in some settings as if it has eight cores.)

In the classification case, one must use a variation of SA. In predicting the class of a new case, each cluster node produces its own prediction. Then “voting” is used: The final predicted class is the one predicted the most often among the r cluster nodes.

The major advantages of SA are its simplicity and generality. One can use it to attain a good speedup on almost any statistical/machine learning algorithm.

Data Input/Output, etc. with `data.table`

As mentioned, `data.table` is an extension of the data frame construct (Dowle 2017). In Big Data contexts, the use of the `data.table` package is indispensable. For most operations – reading

from/writing to disk, data grouping, and so on – it is much faster than the standard R counterparts.

Graphics

Though one might at first think, “The more data, the better,” Big Data can present real problems in terms of graphics. With so many points to plot, there is a real risk of encountering the “black screen problem,” with the points solidly filling major areas of the computer screen, thus rendering the graph meaningless. A number of techniques have been developed to deal with this (Unwin et al. 2007).

In addition, with even moderate values of p , the number of variables/features in our dataset, plotting the data in a visually interpretable way is challenging. Data can readily be visualized in the case $p = 2$, and possibly $p = 3$, but it is difficult for $p > 2$. One method to address this problem is *parallel coordinates* (Inselberg 2009). Here each data point is displayed as a segmented line connecting dots at heights given by the values of the variables in that data point.

Direct use of parallel coordinates with Big Data will typically lead to the “black screen problem.” The `cdparcoord` package (Matloff et al. 2017b; Yang et al. 2017) aims to solve this problem by plotting only the most frequently appearing data tuples.

R

Conclusions

R is a very powerful tool for data science, developed by statisticians, for statisticians. It has truly excellent graphics packages, built-in matrix and linear algebra operations, several types of object-oriented programming models to choose from, and so on. The tremendous repository of contributed packages, CRAN, is a huge advantage by itself.

Care must be taken with speed and memory issues. These are handled by the use of vectorization and taking advantage of several good R facilities for parallel programming, as well as the availability of the `bigmemory` package. Properly used, R is a very strong tool for taming Big Data.

Cross-References

- [Julia](#)
- [Python](#)
- [Parallel Graph Processing](#)
- [Parallel Processing with Big Data](#)

References

- Breshears C (2009) The art of concurrency: a thread monkey's guide to writing parallel applications. O'Reilly Media, Sebastopol
- Bühlmann P, Drineas P, Kane M, van der Laan M (2016) Handbook of big data. Chapman & Hall/CRC handbooks of modern statistical methods. CRC Press, Boca Raton
- Chambers J (2008) Software for data analysis: programming with R. Statistics and computing. Springer, New York
- Chang W (2013) R graphics cookbook. O'Reilly and associate series. O'Reilly Media, Sebastopol, CA
- Dowle M (2017) Data analysis using data.table. <https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html>
- Eddelbuettel D (2013) Seamless R and C++ integration with Rcpp. Use R! Springer, New York
- Inselberg A (2009) Parallel coordinates: visual multidimensional geometry and its applications. Springer, New York
- Kane MJ, Emerson J, Weston S (2013) Scalable strategies for computing with massive data. *J Stat Softw* 55(14):1–19
- Lichman M (2017) UCI machine learning repository. <http://archive.ics.uci.edu/ml>
- Luraschi J, Ushey K, Allaire J (2017) Sparklyr: R interface to Apache Spark. <https://CRAN.R-project.org/package=sparklyr>
- Matloff N (2011) The art of R programming: a tour of statistical software design. No starch press series. No Starch Press, San Francisco
- Matloff N (2015) Parallel computing for data science: with examples in R, C++ and CUDA. Chapman & Hall/CRC the R series. CRC Press, Boca Raton
- Matloff N (2016) Software Alchemy: turning complex statistical computations into embarrassingly-parallel ones. *J Stat Softw* 71(4):1–15
- Matloff N, Fitzgerald C, Davis R, Yancey R, Huang S (2017a) partools: tools for the 'Parallel' package. <https://github.com/matloff/partools>
- Matloff N, Yang V, Nguyen H (2017b) cdparcoord: top frequency-based parallel coordinates. <https://CRAN.R-project.org/package=cdparcoordr>
- Murrell P (2011) R graphics, 2nd edn. Chapman & Hall/CRC the R series. Taylor & Francis, Boca Raton, FL
- Nielsen F (2016) Introduction to HPC with MPI for data science. Undergraduate topics in computer science. Springer International Publishing, Cham
- Plotly Technologies Inc (2015) Collaborative data science. <https://plot.ly>
- R Core Team (2017) R: a language and environment for statistical computing. In: R foundation for statistical computing, Vienna. <https://www.R-project.org/>
- Reinders J (2007) Intel threading building blocks: outfitting C++ for multi-core processor parallelism. O'Reilly series. O'Reilly Media, Sebastopol
- Sarkar D (2008) Lattice: multivariate data visualization with R. Use R! Springer, New York
- Unwin A, Theus M, Hofmann H (2007) Graphics of large datasets: visualizing a million. Statistics and computing. Springer, New York
- Weston S (2017) foreach: provides foreach looping construct for R. <https://CRAN.R-project.org/package=foreach>
- Wickham H (2016) Ggplot2: elegant graphics for data analysis. Use R! Springer International Publishing, New York
- Yang V, Nguyen H, Matloff N, Xie Y (2017) Top-frequency parallel coordinates plots (arxiv). arXiv:1709.00665
- Yu H (2014) [Rmpi] news. <http://www.stats.uwo.ca/faculty/yu/Rmpi/>

Ramp Secret Sharing Scheme (RSSS)

- [Security and Privacy in Big Data Environment](#)

RDF Compression

Miguel A. Martínez-Prieto¹, Javier D. Fernández², Antonio Hernández-Illera¹, and Claudio Gutiérrez³

¹Department of Computer Science, Universidad de Valladolid, Valladolid, Spain

²Complexity Science Hub Vienna, Vienna University of Economics and Business, Vienna, Austria

³Department of Computer Science and Millenium Institute on Foundations of Data, Universidad de Chile, Santiago, Chile

Synonyms

[Linked data compression](#); [Semantic data Compression](#)

Definitions

RDF compression can be defined as the problem of encoding an RDF dataset using less bits than that required by text-based traditional serialization formats like RDF/XML, NTriples, or Turtle, among others. These savings immediately lead to more efficient storage (i.e., archival) and less transmission costs (i.e., less bits over the wire). Although this problem can be easily solved through universal compression (e.g., gzip or bzip2), optimized *RDF-specific compressors* take advantage of the particular features of RDF datasets (such as semantic redundancies) in order to save more bits or to provide retrieval operations on the compressed information. RDF self-indexes are focused on this latter task.

RDF self-indexes are RDF compressors that provide indexing features in a space close to that of the compressed dataset and can be accessed with no prior (or partial) decompression. These properties enhance scalability (i.e., less resources are required to serve semantic data) and speed up access as more information can be managed in higher levels of the memory hierarchy (typically, main memory or cache). In addition, efficient search algorithms have been proposed to resolve basic queries on top of self-indexed datasets. As a result, RDF self-indexes have been adopted as a core component of semantic search engines and lightweight Linked Data servers.

Finally, *RDF stream compressors* specifically focus on compressing a (continuous) stream of

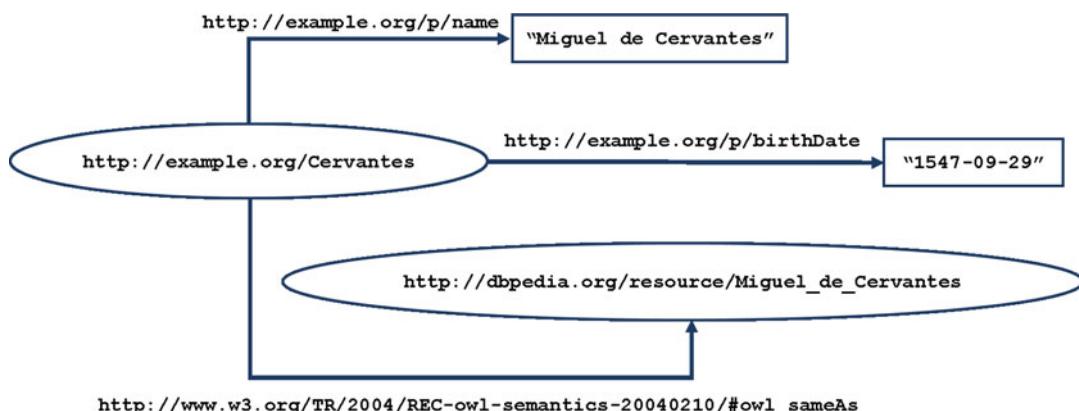
RDF data in order to improve exchange processes, typically in real time. This constitutes a more recent trend that exploits different trade-offs between the space savings achieved by the compressor and the latency introduced in the compression/decompression processes.

This entry introduces basic notions of RDF compression, RDF self-indexing, and RDF stream compression and discusses how existing approaches deal with (and remove) redundant information in semantic datasets.

Overview

RDF (Schreiber and Raimond 2014) stands for *Resource Description Framework*, which allows information about resources (documents, people, physical/logical objects, etc.) to be easily expressed in the form of triples. Each triple comprises the resource being described (referred to as subject), a property of that resource (predicate), and the corresponding value (object). Assuming infinite, mutually disjoint sets U (RDF Uniform Resource Identifiers, *URIs*), B (blank nodes), and L (RDF literals), a triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple (Gutiérrez et al. 2004). A triple (s, p, o) is usually represented as a labeled directed graph: $s \xrightarrow{p} o$.

Figure 1 shows an RDF graph that comprises three triples about the Spanish writer *Miguel de Cervantes*. The node named as [<http://example.org/Cervantes>](http://example.org/Cervantes) declares the corresponding RDF resource, while the labeled edges provide his



RDF Compression, Fig. 1 RDF graph which comprises three triples about *Miguel de Cervantes*

name and *birth date* as literals and a *same as* link pointing to a similar resource <http://dbpedia.org/resource/Miguel_de_Cervantes>, described in DBpedia, a partial conversion of Wikipedia to RDF.

RDF is an extremely simple, but powerful logical model, which can be used to describe and integrate a variety of data from different domains. This flexibility has motivated the adoption of RDF as one of the mainstream semi-structured data model in life sciences, geography, or open-government projects, among other fields of knowledge. Cross-domain datasets, like the aforementioned DBpedia, demonstrate how heterogeneous data can be effectively integrated regardless of its (lack of) structure.

RDF excels at logical level, but important scalability issues arise at physical level. The case of DBpedia is exemplary. The latest edition of this dataset (*DBpedia 2016-10*) consists of roughly 13 billion triples which describe people, places, organizations, films, species, or diseases, among other information available at *Wikipedia*, *Wikipedia Commons*, and *Wikidata*. These descriptions also exploit that the source information is commonly expressed in different languages to consolidate a multilingual RDF-based encyclopedia. The resulting knowledge base is extensively used for multiple purposes and services (e.g., semantic search, entity disambiguation, translation, etc.), and it constitutes a valuable resource for academics and semantic web practitioners. However, although DBpedia can be queried online via different APIs, some applications and services require a complete view of the dataset;

hence, consumers must deal with the high-cost requirements of this huge dataset in terms of space (its current dump needs several hundreds of gigabytes) and the required processing power to operate on this data volume.

These numbers endorse RDF compression as a scalable technique to alleviate the costs of RDF management, which are particularly relevant in a resource-constrained scenario (e.g., light clients or low-performance networks) and Big Semantic Data applications. But, *why are RDF compressors so effective?* Attending to the foundations of data compression, *compression is possible because data is normally represented in a form that is longer than absolutely necessary* (Salomon 2007). This situation is particularly significant in semantic data. An example is depicted in Fig. 2, which shows the previous RDF graph serialized in NTriples (Beckett 2014) and Turtle (Beckett et al. 2014). RDF files are plenty of redundancy, and three different classes are considered (Pan et al. 2014):

Symbolic redundancy is due to symbol repetitions. The main contributors of this significant source of redundancy are the large and highly repetitive URIs used for naming purposes. In practice, an RDF dataset comprises many different URIs, but they are usually defined from a small group of domains and tend to have common long prefixes. Note, for instance, that the prefix <http://example.org/> is shared by three URIs in our example. Some RDF serialization formats, like Turtle, introduce the @prefix constructor to partly address this issue. This allows the original URI to be rewritten as a short reference to the shared prefix (e.g., ns0 or ns1 in Fig. 2)

NTriples

```

<http://example.org/Cervantes> <http://example.org/p/name> "Miguel de Cervantes" .
<http://example.org/Cervantes> <http://example.org/p/birthDate> "1547-09-29" .
<http://example.org/Cervantes> <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/#owl_sameAs>
<http://dbpedia.org/resource/Miguel_de_Cervantes> .

```

Turtle

```

@prefix ns0: <http://example.org/p/> .
@prefix ns1: <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/#> .

<http://example.org/Cervantes>
ns0:name "Miguel de Cervantes" ;
ns0:birthDate "1547-09-29" ;
ns1:owl_sameAs <http://dbpedia.org/resource/Miguel_de_Cervantes> .

```

RDF Compression, Fig. 2 Example of RDF serializations (NTriples & Turtle)

and the remaining suffix. Symbol repetitions are also present in infixes/suffixes of URIs, but their impact is much lower. On the other hand, symbolic redundancy from literals depends on the domain being described, and it can be more difficult to predict. Finally, note that no restrictions are made on blank node serialization, but they also tend to share substrings (as in URIs) (Martínez-Prieto et al. 2012b).

Universal compressors, like gzip or bzip2, remove this class of redundancy and generate files which are suitable for storage and exchange. However, this class of compression prevents other purposes, like random access to the compressed data. *String dictionaries* (Martínez-Prieto et al. 2016) arise as an alternative that provides (slightly) less compression than universal techniques but efficient RDF retrieval operations. Dictionary compression identifies all different strings (*vocabulary*) in an RDF dataset and assigns an integer identifier (ID) to each one (typically an ID $i \in [1, n]$ being n the vocabulary size). This simple decision allows for replacing each term occurrence in the RDF dataset by its corresponding ID, thus shifting from managing potential large strings to small-/medium-size integers. In other words, the original RDF graph is re-encoded as a string dictionary, organizing the vocabulary, and an ID-graph. It is worth noting that RDF vocabularies reach non-negligible sizes and must be also compressed (Martínez-Prieto et al. 2012b).

Syntactic redundancy underlies to the graph-shaped structure of RDF datasets and how this structure is serialized. The simplest serialization formats, like NTriples, write one triple per line and serialize explicitly all its components (e.g., the subject URI is written three times, once per triple, in Fig. 2). In contrast, syntaxes like Turtle introduce subject-based encoding to alleviate this issue, i.e., it allows triples to be encoded as adjacency lists of (predicate, object) pairs related to each subject. Similar observations can be done for predicates and objects. Predicates tend to be repeated across different subjects, but they are often restricted to a domain and range of application, whether explicitly described in a vocabulary (ontology) or implicitly in the data. For example,

the *salary* predicate is not typically related to a *book* but a *person*, while the converse applies to the *pages* property. Grouping and splitting the description of books and persons may save repetitions. In turn, some objects are paired with some particular predicates (e.g., 25 °C is tight to a *temperature* predicate), but these pairs could be reused for different subjects.

Particular *graph compressors* should be used to minimize this class of redundancy. These techniques rearrange graphs in terms of their adjacency information and encode it using list or matrix compression approaches. The most well-known techniques come from web or social graph compression scenarios (Boldi and Vigna 2004), but some of them (Brisaboa et al. 2014) have been tuned to address the particular needs of RDF graphs. It is worth noting that graph compressors often operate on integer representations, so an initial dictionary compression stage is first performed to obtain the corresponding ID-graphs.

Semantic redundancy, in contrast to the previous ones, appears at the logical level. It arises when less triples can be used to provide the same knowledge. Thus, the redundancy does not depend on how triples are encoded, but on the knowledge they provide.

This redundancy cannot be removed using traditional compression approaches. In this case, specific RDF compressors must be designed from scratch in order to obtain the minimal subset of “canonical triples” that allows the original knowledge to be effectively reconstructed. Thus, the compression is achieved by reducing the number of unnecessary encoded triples. It is worth noting that symbolic and syntactic redundancies may be still present on the canonical graph; hence, other forms of compression can be applied on top of the semantic-compressed dataset in order to achieve better compression ratios.

R

Key Research Findings

RDF compression has emerged as an active research and development field over the past years, and some *lossless compressors* (i.e., techniques that can exactly recreate the original

dataset from its compressed representation) have been proposed. RDF compressors can be classified into *physical* and *logical*: the former exploits symbolic/syntactic redundancy, while the latter focuses on semantic-based redundancy. Finally, *hybrid* compressors perform at physical and logical levels.

The simplest form of **physical compression** is *universal compression*, i.e., using any general-purpose technique (e.g., gzip or bzip2) to directly compress an RDF file. This choice is simple and efficient in terms of compression ratio and delays (fast to process), and it can be easily integrated in other workflows as it makes use of standard and widespread techniques. In turn, RDF-specific physical compressors can be designed from scratch in order to deal with specific RDF characteristics and achieve better performance than general-purpose techniques.

RDF-specific physical compressors usually perform *dictionary compression*. That is, they translate the original RDF graph into a new representation which includes a string dictionary and an ID-graph encoding:

- The **dictionary** organizes the RDF vocabulary, which comprises all different terms used in the dataset.
- The **ID-graph** replaces the original terms by their corresponding IDs in the dictionary.

Figure 3 shows a dictionary-compressed representation of the graph example. In this case,

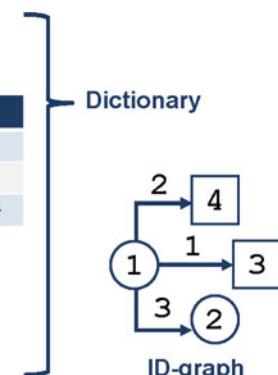
the dictionary comprises three independent mappings by subjects, predicates, and objects, but other configurations are possible. Triples are then encoded as $(1, 1, 2)$ $(1, 2, 3)$ $(1, 3, 1)$, where the original terms can be easily retrieved by using the corresponding mapping; e.g., $(1, 2, 3)$ is translated into:

- $S.get(1) = \langle\text{http://example.org/Cervantes}\rangle$.
- $P.get(2) = \langle\text{http://example.org/p/name}\rangle$.
- $O.get(3) = \text{"Miguel de Cervantes"}$.

Physical compressors propose different approaches to organize and compress RDF dictionaries and to encode the corresponding ID-graph representations.

Dictionary compression has not received much particular attention, in spite that representing the RDF vocabulary usually takes more space than the ID-graph encoding (Martínez-Prieto et al. 2012b). As in our example, RDF vocabularies can be intuitively partitioned by term roles (subjects, predicates, and objects). However, this approach is not totally effective because terms that perform subject and object roles are encoded twice, and these terms can sum up to 60% of the vocabulary terms (Martínez-Prieto et al. 2012b). Thus, a four-vocabulary configuration (Atre et al. 2010) is massively used, where the dictionary holds independent mappings for (i) shared *subjects-objects*, (ii) single *subjects* (not occurring as objects), and (iii) single *objects* (not occurring as subjects) and *predicates*.

ID	Subject Term	
1	<code><http://example.org/Cervantes></code>	
ID	Predicate Term	
1	<code><http://example.org/p/birthDate></code>	
2	<code><http://example.org/p/name></code>	
3	<code><http://www.w3.org/TR/2004/REC-owl-semantics-20040210/#owl_sameAs></code>	
ID	Object Term	
1	<code><http://dbpedia.org/resource/Miguel_de_Cervantes></code>	
2	<code>"1547-09-29"</code>	
3	<code>"Miguel de Cervantes"</code>	



RDF Compression, Fig. 3 Dictionary-compressed representation

Vocabularies are further splitted by URIs, blank nodes, and literals for choosing the best compression technique for each collection of RDF terms. Martínez-Prieto et al. (2016) propose different approaches to compress these RDF vocabularies. URI (and blank node) vocabularies can be effectively compressed using Front-Coding, a differential encoding mechanism that exploits shared long prefixes and reduces URI space requirements up to 20 times. On the other hand, literal vocabularies are, in general, less predictable, hampering their compression. Thus, dictionaries that combine grammar-based compression with Front-Coding or hashing can report huge space savings of up to six to ten times the original space. Besides space reductions, these dictionary compression techniques must be able to efficiently translate IDs into the corresponding RDF terms, ensuring efficient *decoding* performance, most of them working at microsecond level. These techniques often support the inverse *search* functionality to translate an RDF term into its corresponding ID, which is essential for RDF self-indexing. In addition, they can provide *prefix*- and *substring*-based retrieval of RDF terms.

Once removed symbol repetitions, *ID-graph compression* looks for syntactic redundancy on the resulting ID-graph. These techniques model the graph in terms of adjacency lists or matrices, and look for regularities or patterns, which are succinctly encoded. HDT (Fernández et al. 2013) proposes *BitmapTriples*, one of the pioneer approaches for (RDF) ID-graph compression. In essence, it transforms the graph into a forest of three-level trees: each tree is rooted by a subject ID, having its adjacency list of predicates in the second level and, for each of them, the adjacency list of related objects in the third (leaf) level. The whole forest is then compressed using two ID sequences (for predicates and objects) and two bit sequences which encode the number of branches and leaves of each tree. This simple encoding reports interesting compression ratios (10–25% of the original space), while supporting efficient triple decoding. Furthermore, BitmapTriples allows subject-based queries to be resolved by traversing subject trees from the root. HDT con-

solidates a binary serialization format by joining Front-Coding and BitmapTriples to compress dictionaries and ID-graphs, respectively.

OFR (Swacha and Grabowski 2015) proposes another compression scheme for ID-graphs. It first performs dictionary compression (terms are organized into a multi-dictionary using differential encoding), and the resulting ID-graph is sorted by objects and subjects. In this case, *run-length* and *delta* compression (Salomon 2007) are applied to exploit multiple object occurrences, and the non-decreasing order of the consecutive subjects, respectively. OFR compressed files are then recompressed using universal techniques like *gzip* or *7zip*. The resulting OFR effectiveness improves HDT, but its inner data organization discourages any chance of efficient retrieval.

Efficient RDF retrieval is addressed by RDF self-indexes. These approaches do not just compress the ID-graph, but also provide indexing capabilities over it. HDT-FoQ (Martínez-Prieto et al. 2012a) enhances HDT to also support predicate- and object-based queries, adding inverted indexes for predicate and object adjacency lists that, all together, provide excellent performance for resolving SPARQL triple patterns. k^2 -triples (Álvarez-García et al. 2014) provides an alternative organization of the ID-graph, encoding a (binary) adjacency matrix of (subject, object) pairs per predicate. These matrices are very sparse and can be easily compressed using k^2 -trees (Brisaboa et al. 2014). The k^2 -triples approach improves HDT-FoQ compression ratios, and reports competitive numbers for all triple patterns binding the predicate, but results in a poor performance in those queries with unbounded predicates. This is mitigated by adding two additional indexes to store the predicates related to each subject and object, but the pattern that only binds the predicate remains slow. RDFCSA (Brisaboa et al. 2015) is the most recent RDF self-index, which encodes the ID-graph as a compressed suffix array (CSA). RDFCSA also ensures efficient lookup performance, competing with k^2 -triples at the cost of using more space.

Logical compressors look for (redundant) triples that can be inferred from others. These

triples are removed from the original graph, and only the resulting *canonical subgraph* is finally serialized. Different approaches have been followed to obtain these canonical subgraphs. The initial approaches (Iannone et al. 2005; Meier 2008) are based on the notion of *lean subgraph*. The lean subgraph is a subset of the original graph that has the property of being the smallest subgraph that is instance of the original graph. The number of removed triples by a lean subgraph strongly depends on the graph features, but a reasonable lower limit is two removed triples per blank node (Iannone et al. 2005). Nevertheless, some triples of a lean graph can still be derived from others; hence, some semantic redundancy can still be present (Meier 2008).

The rule-based (RB) compressor (Joshi et al. 2013) uses mining techniques to detect objects that are commonly related to a particular predicate (intra-property patterns) and to group frequent predicate-object pairs (inter-property patterns). These patterns are then used as generative rules to remove triples that can be inferred from such patterns. RB is not so effective by itself, and only inter-property patterns enable significant amount of triples to be removed. Venkataraman and Sreenivasa Kumar (2015) state that frequent patterns are not so expressive to capture semantic redundancy and suggest that effectiveness can be improved using more expressive rules. In this case, Horn rules are mined from the dataset, and all triples matching their head part are removed. The resulting canonical subgraph is then compressed using RB. This Horn-rule-based compressor outperforms RB effectiveness, but it introduces latencies in compression and decompression processes.

Hybrid compressors compact the RDF graph by first using a logical approach to remove redundant triples and then performing physical compression at serialization level. Although these techniques could combine the best of logical and physical compression, their application has received relatively little attention until now.

HDT++ (Hernández-Illera et al. 2015) revisits HDT to introduce some methods to detect syntactic and semantic redundancy. HDT++ brings out

the inherent structure of RDF by detecting and grouping the different set of predicates (*predicate families*) used to describe subjects. The original RDF graph is encoded as a set of subgraphs, one per predicate family. The `rdf:type` values are attached to each predicate family, hence removing these triples from the subgraphs. Finally, HDT++ uses local IDs for the terms in each subgraph, thus reducing the number of required bits. As a result, HDT++ reduces the original HDT ID-graph space requirements up to two times for more structured datasets and reports significant improvements even for highly semi-structured datasets.

The *graph pattern-based* (GPB) compressor (Pan et al. 2015) shares some common features with HDT++, also grouping subjects by predicate families, called *entity description patterns* (EDPs). Each EDP is encoded as a pair which includes the corresponding pattern and all instances matching it. This policy consolidates the simplest GPB encoding scheme (LV0), but patterns are then merged to obtain better patterns (LV1), and the merging process can be recursively performed (LV2). GPB results are not compared with other physical compressors, but they excel at logical level, where GPB-LV2 is able to remove more triples than RB.

Finally, *RDF2NormRDF* (Ticona-Herrera et al. 2015) is not a compressor, but an approach to normalize RDF data. It applies different transformation rules to remove duplicated edges and nodes and also deals with particular blank node features. At physical level, RDF2NormRDF focuses on namespace issues and normalizing types and language tags encodings. As expected, RDF2NormRDF does not generally remove more redundancy than other compressors, such as HDT, but it reports better compression ratios for small datasets.

Real-Time Compression

All previous compressors share a common feature: they are designed to perform in batch processing scenarios where time requirements are not so strict. That is, the target system can wait a reasonable time to receive the compressed data, and the compression process is performed *offline*.

However, more and more systems perform *online* data processing (in real time), where delays must be minimized. In these cases, the volume of RDF is traditionally smaller, but triples are continuously generated, and these semantic streams must be efficiently distributed over a network.

Thus, a key challenge for RDF stream processing systems is the ability to consume increasingly large volumes of data with varying and potentially high input rates. This scenario also claims for RDF compression that, in this case, put the focus on minimizing the elapsed time since the original piece of data is available at the producer, until it is ready to be used at the consumer. This process encompasses three tasks: (i) *data compression* (at the producer), (ii) *data transmission* over a network, and (iii) *data decompression* (at the consumer). The real challenge for RDF stream compressors is finding the right space/time trade-off which optimizes the overall workflow performance. RDF-specific compression over streaming data is usually performed at *physical level* and mostly as adaptations of existing RDF compressors.

Streaming HDT (Hasemann et al. 2012) adapts HDT to simplify the associated metadata and restrict the number of terms managed by the dictionary; hence, shorter IDs are used. The approach is then tailored to constrained devices, where the vocabulary of terms is very limited. In turn, RDSZ (Fernández et al. 2014a) uses differential encoding to take advantage of the similarities between consecutive triples sent over the wire. In addition, the resultant scheme is compressed with Zlib to exploit additional redundancies. Overall, RDSZ gains in compression (17% on average) are at the cost of increasing the processing time.

ERI (Fernández et al. 2014b) is an RDF stream compressor that adapts the W3C Efficient XML Interchange (EXI) format (Schneider et al. 2014) for RDF data. Similarly to HDT++ and GPB, ERI tries to detect and encode the predicate families but in a dynamic fashion. In addition, highly repeated object values (besides the values for *rdf:type*) are also encoded in the families. Then, the non-repeated values for each particular predicate in the family are encoded in a channel,

where specific compression can be applied (the concept is then similar to leveraging the locality in column-based databases). As a result, ERI achieves a slightly better compression than general-purpose stream-enabled compressors (such as Zlib) with limited latency overhead. Note that the XML-based compression of EXI can also be adapted to RDF streams by (i) forcing the serialization format of RDF to RDF/XML and, optionally, (ii) generating an application-specific grammar that encodes the repeated values.

The pattern-based approach of ERI has been also followed by two recent techniques, PatBin (Lhez et al. 2017) and FSSD (Karim et al. 2017). PatBin approach performs dictionary-based compression and then splits the graph in ID-patterns (essentially predicate families) and value/variable bindings. The Factorizing Semantic Sensor Data (FSSD) technique uses a deductive database system to encode the repeated predicate families and object values as datalog rules, which are then applied on the input data to achieve important size reductions. However, although FSSD has been tested on sensor data, the factorization is still performed in an offline fashion. In general, finding patterns in RDF Streams efficiently can be seen as an orthogonal approach.

Examples of Application

R

RDF compression has been widely adopted by the Semantic Web community as a standard technique to reduce storage and transmission costs when downloading RDF datasets. Most of the publishers in the Linked Open Data (LOD) cloud make use of universal compression, given its simplicity, usability, and widespread adoption. This is particularly true for projects publishing massive amounts of RDF data, such as DBpedia or Bio2RDF.

Nonetheless, RDF-specific compressors, and in particular RDF self-indexes, are receiving increased attention. Projects like LOD Laundromat (Beek et al. 2014) or Triple Pattern Fragments (TPF) (Verborgh et al. 2016) describe two in-

teresting use cases exploiting compressed RDF. LOD Laundromat is an initiative to crawl and clean (removing syntax errors) RDF data from the LOD cloud. As a result, it exposes more than 650 K cleansed datasets which are delivered in HDT format and can be queried using TPF interfaces. TPF focuses on alleviating the burden of endpoints by serving simple SPARQL triple patterns, paginating the results. This simplification allows servers to scale, while clients can always execute more complex SPARQL queries on top of TPFs by taking care of integrating and filtering the results. Given the simplicity of the required infrastructure at the server, TPF interfaces can make use of RDF self-indexes to serve low-cost operations, being HDT the most used backend in practice. The recently published *LOD-a-lot* dataset (Fernández et al. 2017) combines the benefits from both projects to provide a practical example of efficient management of compressed Big Semantic Data. *LOD-a-lot* integrates all data from LOD Laundromat into a cross-domain mashup of more than 28 billion triples and several terabytes of space (in NTriples). This dataset is then exposed as HDT and the corresponding TPF interface. The queryable self-indexed HDT of such large portion of the LOD cloud takes 524 GB and can serve fast triple pattern resolution with an affordable memory footprint (in practice, 15.7 GB). These numbers are a strong evidence of how RDF compression contributes to make Big Semantic Data management feasible in most Linked Data servers (for online consumption) and clients (for downloading and offline consumption).

RDF compression and self-indexes have also been actively used in other Semantic Web areas such as (i) SPARQL querying and recommender systems (Martínez-Prieto et al. 2012a; Heitmann and Haye 2014), leveraging the retrieval operations supported by self-indexes to support more complex queries; (ii) reasoning (Cure et al. 2015), optimizing the RDF dictionary and triples encoding to serve inference capabilities; (iii) versioned RDF or RDF archives (Fernández et al. 2016), where RDF compression is used to preserve (and query) the history of an RDF dataset; and (iv) constrained and mobile devices (Käbisch et al.

2015) in order to maximize the exploitation of their storage/processing capabilities.

Finally, RDF compression has also been highlighted by RDF stream processing systems: CQELS Cloud (Le-Phuoc et al. 2013) uses a basic dictionary-based approach to process and move IDs (integers) between nodes, and Zstreamy (Fisteus et al. 2014) exploits the Zlib compressor with similar purposes. A last trend regards querying on compressed RDF streams (without prior decompression). A recent approach (Déme et al. 2017) extends RDFSZ to resolve basic SPARQL queries, as well as filters and aggregations.

Future Directions for Research

The state of the art of RDF compression shows many and varied techniques which exploit symbolic, syntactic, and semantic redundancy from different perspectives. They (i) save much storage space, (ii) reduce network latencies, or (iii) improve RDF retrieval performance, among other achievements. Nonetheless, RDF compression still remains an open challenge. We can categorize future directions in three categories, *low-level* optimization, *scalable management*, and *applications*.

First, physical and logical compression should be independently explored to determine more effective approaches which, desirably, should be plugged into powerful hybrid compressors. These achievements should be aligned with recent advances in succinct data structures and self-indexes, which try to squeeze the space requirements while providing additional retrieval operations. Querying compressed RDF is a main direction for research to consolidate scalable and efficient semantic search engines. These additional *low-level* optimizations can boost the adoption of RDF compression and self-indexing, which is already at the edge of conforming a core component of scalable Semantic Web applications.

In general, RDF compression has already proved its ability for storing, exchanging, processing, and querying Big Semantic Data.

However the *scalable management* of compressed RDF datasets has room for optimization. On the one hand, the compression of Big Semantic Data, as well as the creation of RDF self-indexes, suffers from scalability problems, as they can require high amounts of memory and processing power. Although this process is a one-off task in many scenarios, it can introduce unaffordable costs for publishers. Current efforts focus on exploring computation models like MapReduce or Spark to alleviate this burden, yet additional research must be conducted to consolidate a scalable approach for Big Semantic Data compression. On the other hand, most of RDF self-indexes are mainly static or costly to update; hence, dynamic techniques for succinct data structures and self-indexes are an active area of research.

Finally, RDF compression and its *applications* have many unexplored directions. Adoption of RDF compressed techniques in combination with existing triple stores and reasoners, integration within semantic data warehouses, resolution of temporal queries on compressed RDF archives, or integration within natural language processing tasks (e.g., entity discovering and linking) are only some prominent applications which claim for innovative RDF compression techniques to scale in a Big Semantic Data scenario.

Cross-References

- ▶ [RDF Serialization and Archival](#)
- ▶ [Semantic Search](#)

References

- Álvarez-García S, Brisaboa N, Fernández JD, Martínez-Prieto MA, Navarro G (2014) Compressed vertical partitioning for efficient RDF management. *Knowl Inf Syst* 44(2):439–474
- Atre M, Chaoji V, Zaki M, Hendler J (2010) Matrix “Bit” loaded: a scalable lightweight join query processor for RDF data. In: 19th international conference on World Wide Web (WWW), pp 41–50
- Beckett D (2014) RDF 1.1 N-Triples. W3C recommendation. <https://www.w3.org/TR/2014/REC-n-triples-20140225/>
- Beckett D, Berners-Lee T, Prud'hommeaux E, Carothers G (2014) RDF 1.1 turtle. W3C recommendation. <https://www.w3.org/TR/2014/REC-turtle-20140225/>
- Beek W, Rietveld L, Bazoobandi HR, Wilemaker J, Schlobach S (2014) LOD Laundromat: a uniform way of publishing other people's dirty data. In: 13th international semantic web conference (ISWC), pp 213–228
- Boldi P, Vigna S (2004) Webgraph framework I: compression techniques. In: 13th international conference on World Wide Web (WWW), pp 595–602
- Brisaboa N, Ladra S, Navarro G (2014) Compact representation of web graphs with extended functionality. *Inf Syst* 39(1):152–174
- Brisaboa N, Cerdeira-Pena A, Fariña, Navarro G (2015) A compact RDF store using suffix arrays. In: 22nd international symposium on string processing and information retrieval (SPIRE), pp 103–115
- Cure O, Naacke H, Randriamalala T, Amann B (2015) LiteMat: a scalable, cost-efficient inference encoding scheme for large RDF graphs. In: 2015 IEEE international conference on big data (Big Data), pp 1823–1830
- Déme NB, Dia AF, Boly A, Kazi-Aoul Z, Chiky R (2017) An efficient approach for real-time processing of RDSZ-based compressed RDF streams. In: 15th international conference on software engineering, management and applications (SERA), pp 147–166
- Fernández JD, Martínez-Prieto MA, Gutiérrez C, Polleres A, Arias M (2013) Binary RDF representation for publication and exchange. *J Web Semant* 19:22–41
- Fernández N, Arias J, Sánchez L, Fuentes-Lorenzo D, Corcho Ó (2014a) RDSZ: an approach for lossless RDF stream compression. In: 11th European conference on the semantic web (ESWC), pp 52–67
- Fernández JD, Llaves A, Corcho O (2014b) Efficient RDF interchange (ERI) format for RDF data streams. In: 13th international semantic web conference (ISWC), pp 244–259
- Fernández JD, Umbrich J, Polleres A, Knuth M (2016) Evaluating query and storage strategies for RDF archives. In: 12th international conference on semantic system (SEMANTiCS), pp 41–48
- Fernández JD, Beek W, Martínez-Prieto MA, Arias M (2017) LOD-a-lot – a queryable dump of the LOD cloud. In: 16th international semantic web conference (ISWC), vol 2, pp 75–83
- Fisteus JA, Fernández García N, Sánchez Fernández L, Fuentes-Lorenzo D (2014) Zstreamy: a middleware for publishing semantic streams on the web. *J Web Semant* 25:16–23
- Gutiérrez C, Hurtado C, Mendelzon AO (2004) Foundations of semantic web databases. In: 23rd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS), pp 95–106
- Hasemann H, Kroller A, Pagel M (2012) RDF provisioning for the Internet of things. In: 3rd international conference on the Internet of things (IOT), pp 143–150
- Heitmann B, Haye C (2014) SemStim at the LOD-RecSys 2014 challenge. In: Semantic web evaluation challenge (SemWebEval), pp 170–175

- Hernández-Illera A, Martínez-Prieto MA, Fernández JD (2015) Serializing RDF in compressed space. In: 21st data compression conference (DCC), pp 363–372
- Iannone L, Palmisano I, Redavid D (2005) Optimizing RDF storage removing redundancies: an algorithm. In: 18th international conference on industrial and engineering applications of artificial intelligence and expert systems (IEA/AIE), pp 732–742
- Joshi A, Hitzler P, Dong G (2013) Logical linked data compression. In: 10th extended semantic web conference (ESWC), pp 170–184
- Käbisch S, Peintner D, Anicic D (2015) Standardized and efficient RDF encoding for constrained embedded networks. In: 12th European conference on the semantic web (ESWC), pp 437–452
- Karim F, Vidal ME, Auer S (2017) Efficient processing of semantically represented sensor data. In: 13th international conference on web information systems and technologies (WEBIST), pp 252–259
- Le-Phuoc D, Quoc HNM, Le Van C, Hauswirth M (2013) Elastic and scalable processing of linked stream data in the cloud. In: 12th international semantic web conference (ISWC), pp 280–297
- Lhez J, Ren X, Belabbess B, Curé O (2017) A compressed, inference-enabled encoding scheme for RDF stream processing. In: 14th European conference on the semantic web (ESWC), pp 79–93
- Martínez-Prieto MA, Arias M, Fernández JD (2012a) Exchange and consumption of huge RDF data. In: 9th extended semantic web conference (ESWC), pp 437–452
- Martínez-Prieto MA, Fernández JD, Cánovas R (2012b) Compression of RDF dictionaries. In: 27th ACM international symposium on applied computing (SAC), pp 1841–1848
- Martínez-Prieto M, Brisaboa N, Cánovas R, Claude F, Navarro G (2016) Practical compressed string dictionaries. Inf Syst 56:73–108
- Meier M (2008) Towards rule-based minimization of RDF graphs under constraints. In: 2nd international conference on web reasoning and rule systems (RR), pp 89–103
- Pan J, Gómez-Pérez J, Ren Y, Wu H, Zhu M (2014) SSP: compressing RDF data by summarisation, serialisation and predictive encoding. Technical report. http://www.kdrive-project.eu/wp-content/uploads/2014/06/WP3-TR2-2014_SSP.pdf
- Pan J, Gómez-Pérez J, Ren Y, Wu H, Haofen W, Zhu M (2015) Graph pattern based RDF data compression. In: 4th joint international conference on semantic technology (JIST), pp 239–256
- Salomon D (2007) Data compression: the complete reference. Springer, New York
- Schneider J, Kamiya T, Peintner D, Kyusakov R (2014) Efficient XML interchange (EXI) format 1.0. W3C recommendation
- Schreiber G, Raimond Y (2014) RDF 1.1 primer. W3C working group note. <https://www.w3.org/TR/rdf11-primer/>
- Swacha J, Grabowski S (2015) OFR: an efficient representation of RDF datasets. In: 4th symposium on languages, applications and technologies (SLATE), pp 224–235
- Ticona-Herrera R, Tekli R, Chbeir J, Laborie S, Dongo I, Guzman R (2015) Toward RDF normalization. In: 34th international conference on conceptual modeling (ER), pp 261–275
- Venkataraman G, Sreenivasa Kumar P (2015) Horn-rule based compression technique for RDF data. In: 30th annual ACM symposium on applied computing (SAC), pp 396–401
- Verborgh R, Vander Sande M, Hartig O, Van Herwegen J, De Vocht L, De Meester B, Haesendonck G, Colpaert P (2016) Triple pattern fragments: a low-cost knowledge graph interface for the web. J Web Semant 37–38: 184–206

RDF Dataset Profiling

Stefan Dietze¹, Elena Demidova¹, and Konstantin Todorov²

¹L3S Research Center, Leibniz Universität Hannover, Hanover, Germany

²LIRMM, University of Montpellier, Montpellier, France

Definitions

In the context of this chapter, *an RDF dataset* is defined in accordance with the dataset definition in the Vocabulary of Interlinked Datasets (VoID), (<http://vocab.deri.ie/void>), namely, “*A Dataset* is a set of RDF triples that are published, maintained or aggregated by a single provider.” According to VoID, a dataset represents a meaningful collection of triples as envisioned by its provider. *An RDF dataset profile* is a formal representation of a set of dataset characteristics (features). It describes the dataset and aids dataset discovery, recommendation, and comparison with regard to the represented features. A *dataset profile feature* is a characteristic describing a certain attribute of the dataset. For instance, “*dataset conciseness*” is a dataset profile feature providing information on the degree of redundancy of the information contained in the dataset. A dataset profile is extensible with respect to the

features it contains. Usually, the relevant feature set is application-oriented and depends on the envisaged application scenarios.

Overview

A number of popular dataset registries have emerged, which tackle the problem of dataset discovery through the curation of lightweight dataset descriptions, often also exposing structured metadata according to the state-of-the-art vocabularies such as DCAT (<http://www.w3.org/TR/vocab-dcat/>) or VoID. Popular examples include DataHub (<http://www.datahub.io>) or DataCite (<https://www.datacite.org/>), while the LinkedUp Catalog (<http://data.linkededucation.org/linkedup/catalog/>) (for education) represents a domain-specific example. However, while such metadata is usually edited and curated manually, it is often sparse, not in sync with the constant evolution of the actual datasets, and prone to errors.

On the one hand, as the Web of Data as a whole is evolving along with the constant evolution of individual datasets, manual assessment and representation of a large variety of dataset features is neither feasible nor sustainable. On the other hand, a wide variety of competing as well as complementary approaches exist, aimed at automatic assessment and description of arbitrary datasets. This body of work is spanning several research communities and includes works in fields such as *dataset characterization*, *data summarization*, *dataset assessment*, or *dataset profiling*. While the problem of dataset profiling is of particular importance in the context of the Web of Data, it has been identified and approached already in other related fields, such as general database and data management research.

Emerging from the aforementioned works, a wealth of tools, methods, vocabularies, and applications for assessing, describing, and profiling datasets has become available throughout the past few years, where Ben Ellefi et al. (2017) provides an initial overview and classification.

The aim of this chapter is to provide researchers, dataset providers, and application de-

velopers with an overview of *dataset profiling* and closely related approaches, including *dataset profile features*, *feature extraction methods and tools*, *vocabularies*, and example *applications* to encourage experimentation and facilitate the broader use of RDF datasets.

Key Findings

In order to provide an overview of RDF dataset profiling, we structure key findings into types of *features*, related *methods and tools*, *vocabularies* for their representation, and actual *applications*.

Features, Methods, and Tools

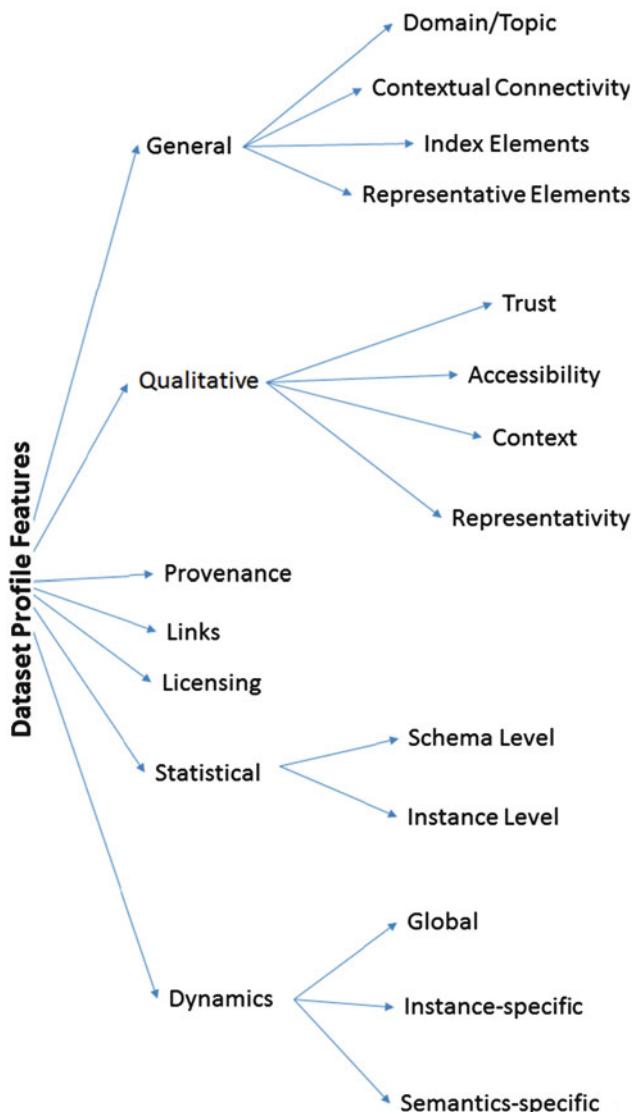
This section provides an inventory of dataset features of relevance for dataset profiling, organized in an extensible feature taxonomy, depicted in Fig. 1. This taxonomy reflects the authors' consensus and provides one of several possible categorization systems. In particular, the features are organized into the following top-level categories: *General*, *Qualitative*, *Provenance*, *Links*, *Licensing*, *Statistical*, and *Dynamics*. This taxonomy guides the categorization of the extraction methods and tools, of which we provide examples for each category. The taxonomy also serves as a backbone for the classification of profiling vocabularies and applications, introduced in the following section. For a complete overview, the reader is referred to the survey (Ben Ellefi et al. 2017).

R

General features. Features in this category carry high-level semantic information, including the *domain* (field of knowledge) or *topic* of the dataset; its *contextual connectivity*, understood as the extensional and/or topical overlap with other datasets; the *index elements* that are pointing to the dataset in a given index or a set of indices; as well as the dataset *representative elements*, i.e., the sets of most descriptive types, properties, or instance samples to characterize the entire dataset.

Extraction Methods and Tools. General features typically require domain knowledge with respect to the content of the dataset. As a best

RDF Dataset Profiling,
Fig. 1 A taxonomy of
dataset profile features



practice, these features should be provided by the data domain experts (e.g., data providers or maintainers) to ensure a high-quality profile. The *topic* and the *contextual connectivity* of the dataset can be described through named entities (NE) extracted from the literal values, as well as categories and clusters of these entities. Furthermore, RDF dataset profiles can include selected *index elements* from the instance-level and schema-level indices or data summaries such as a QTree (Harth et al. 2010). *Representative elements* of an RDF dataset can be the entities with thorough descriptions or predicates obtained

using key discovery approaches. Tools in this category include Linked Data Observatory (Fetahu et al. 2014) and voiDge (Böhm et al. 2011).

Qualitative features. The *Qualitative* features category groups together a number of features known from the long-going study of data quality in computer science in general and the Web of Data in particular (Zaveri et al. 2016; Bizer and Cyganiak 2009). These features are organized into the following subcategories: (1) *Trust*, (2) *Accessibility*, (3) *Representativity*, and (4) *Context/Task Specificity*. *Trust* features relate to the

verifiability of information, its correctness or believability, and the reputation of the publisher and that of the dataset. *Accessibility* regards the facility of access to information and the versatility of access methods, but also the degree of security of information transactions, as well as the access performance.

The features in the *Representativity* group provides information in terms of completeness, understandability, semantic accuracy, conciseness (or redundancy), and consistency (i.e., presence of contradictory information).

Finally, the *Context/Task Specificity* group includes features that measure data quality with respect to a specific task.

Extraction Methods and Tools. For the features in the *Accessibility* category, assessment methods include: (1) exploiting dataset metadata and user annotations to assess *trust*; (2) querying SPARQL endpoints and analysing URIs to measure *availability*; (3) analyzing use of digital signatures and provenance information to assess *security*; (4) assessing latency and scalability in response to the user requests to measure *performance*; and (5) analyzing available serialization formats and languages to assess *versatility*.

Representativity features can be assessed through: (1) exploiting statistical distributions to measure *completeness*; (2) detection of human-readable labeling, metadata availability, as well as communication channels to assess *understandability*; (3) statistical methods and crowdsourcing to measure *semantic accuracy/correctness*; (4) measuring the proportion of unique elements at the schema and instance level to assess *conciseness*; and (5) verification of the schema definitions, correctness of the schema usage, and identification of data inconsistencies (e.g., through semantic constraints) to measure *consistency*.

Context features can be assessed through relevance, sufficiency, and timeliness (e.g., the age of the data and its temporal validity as it is delivered to the user) for a specific task or query.

Example tools to generate qualitative features in several categories include WIQA (Bizer and Cyganiak 2009) and ODPW (Umbrich et al. 2015).

Provenance features. These features are seen as contextual metadata that provide indicators about the origin, timeliness, currency, and update cycles of datasets. These are important characteristics that allow to understand the origins of data, to trace errors, and, ultimately, to establish trust.

Extraction Methods and Tools. *Provenance features* can be extracted from the annotations specified by the data provider via provenance vocabularies using tools like WIQA and ODPW.

Links features. *Links features* are understood as either the number of datasets, with which a dataset is interlinked, or the number of triples in a dataset, in which the subject and the object refer to different datasets. We distinguish between (i) *explicit links* (when datasets have linked instances via owl : sameAs statements) and (ii) *implicit links* (when datasets share topics or contexts, expressed, e.g., by an rdfs : seeAlso statement).

Extraction Methods and Tools. Features related to schema-level and instance-level links can be accessed through the number of interlinked external instances or datasets. These features can be extracted by tools like voiDge (Böhm et al. 2011).

Licensing features. *Licensing features* comprise the type of license under which a dataset is published, indicating whether reproduction, distribution, modification, or redistribution is permitted.

Extraction Methods and Tools. Features in the licensing group are meant to be augmented manually by the data provider and can be collected from the dataset metadata using tools like ODPW.

Statistical features. This type of features may refer to the size of a dataset, its coverage, average number of triples, property co-occurrence, and others. These characteristics can be measured at the *schema level* (class/properties usage count (in general, per subject and per object) or class/properties hierarchy depth) and at the *instance level* (URI usage per subject (/object), triples having a resource (/blanks) as subject (/object), triples with literals, min(/max/avg.) per data type, etc.).

Extraction Methods and Tools. Features from this group can be extracted automatically by applications like LODStats using statement-streaming approaches (Auer et al. 2012). Example statistics include frequencies of vocabulary usage, average length of literals, and number of namespaces used. ProLOD++ (Abedjan et al. 2014) supports different granularities of statistics using clustering approaches prior to the statistics generation.

Dynamics features. This category reflects the dynamic aspects of a dataset and includes features in the global, instance-specific, and semantics-specific categories. In principle, every dataset feature can be dynamic, i.e., changing over time. However, the dynamic aspects of a dataset are considered as separate features describing data, while nonetheless acknowledging their transversal nature (the fact that they span over several of the feature categories described above). These features are organized into three subcategories. *Global* features concern dynamicity aspects related to life span, stability (as an aggregation of the stability of multiple dataset characteristics), and history of update (frequency, degree, and patterns of change). *Instance-specific features* refer to the level of growth of the entities in a dataset, the stability of the identifiers (URIs), and the links between entities. Finally, *semantic-specific* features reflect changes related to structure, domain, and vocabulary.

Extraction Methods and Tools. Global dynamic features can be extracted automatically from the SPARQL endpoints via tracking changes using broadcasting or query techniques as well as by regular crawling of LOD data sources and statistical analysis of the crawls, e.g., in Dyldo (Käfer et al. 2013). Instance-specific dynamic features include observation of resource referencing through notification services, monitoring of links stability, and time-travel tools to access archived representations of the URIs. Note that the extraction of dynamic features, as well as statistical, qualitative, and links features, would in general require

less domain expertise and can be extracted automatically by applications in many cases. To cope with the large scale of data during automatic feature extraction, various sampling techniques can be applied (Fetahu et al. 2014).

Vocabularies

Vocabularies for representing dataset profiles range from general dataset metadata vocabularies to specific vocabularies aimed at representing particular features or feature categories.

The Vocabulary of Interlinked Datasets (VoID) (Alexander et al. 2009) provides a core vocabulary for describing datasets and their links, following a similar rationale as the Data Catalog (DCAT) vocabulary (<http://www.w3.org/TR/vocab-dcat/>), which is partly derived from Dublin Core. Both are widely used to generate basic profiles (Ben Ellefi et al. 2017) and are commonly extended with more feature-specific vocabularies.

With respect to *quality*, the dataset quality (daQ) vocabulary (<http://purl.org/eis/vocab/daq>) (Debattista et al. 2014) and the Data Quality Vocabulary (DQV) (<https://www.w3.org/TR/vocab-dqv/>) provide complementary terms for annotating DCAT dataset descriptions with quality aspects and metrics. Fürber and Hepp (2011) describes the DQM Ontology (<http://semwebquality.org/dqm-vocabulary/v1/dqm>), a general vocabulary for representing data quality features.

Regarding *provenance*, voidp (Omitola et al. 2011) builds on and extends VoID to describe the provenance relationships of data across linked datasets, while the *provenance vocabulary* (<http://trdf.sourceforge.net/provenance/ns.html>) was developed to describe provenance of Linked Data.

From the perspective of archiving and long-term preservation of data, the *Data Dictionary for Preservation Metadata (PREMIS)* (<http://bit.ly/premisOntology>) set of terms can be used to describe the provenance of archived, digital objects (e.g., files, bitstreams, aggregations, and datasets).

Most notably, the *PROV Ontology (PROV-O)* (<http://www.w3.org/TR/prov-o/>) was published

as a W3C Recommendation to be a new quasi-standard for representing provenance and is part of a larger *PROV Family of Documents* (Missier et al. 2013) created to support “the widespread publication and use of provenance information of Web documents, data, and resources.”

Links as important features of Linked Data datasets are represented through a variety of means, covering both schema-level and entity-level links, e.g., instantiating VoID linksets or using *SKOS* (<https://www.w3.org/TR/2009/REC-skos-reference-20090818/>) as a formal vocabulary for defining taxonomic and mapping relations among both concepts and entities. In addition, the Expressive and Declarative Ontology Alignment Language (EDOAL) (<http://alignapi.gforge.inria.fr/edoal.html>) enables the representation of correspondences between entities and concepts in different datasets beyond mere mapping relationships (equivalence, subsumption) using complex formalisms.

General resource metadata vocabularies provide basic features to indicate licensing information, including the *DCMI Metadata Terms* (<http://dublincore.org/documents/dcmi-terms>), featuring dedicated *license* and *rights* properties.

These are complemented through dedicated *licensing* vocabularies, such as ccREL (REL, or rights expression language) vocabulary (https://wiki.creativecommons.org/wiki/CC_REL), which facilitates the representation of Creative Commons licenses in RDF. Similarly, the *Open Digital Rights Language (ODRL)* vocabulary (<http://www.w3.org/community/odrl/two/model/>) enables the fine-grained specification of licensing terms (rights, policies, etc.) in a machine-readable format.

To support the representation of dataset *statistics*, vocabularies such as the RDF Data Cube vocabulary (<http://www.w3.org/TR/vocab-data-cube>), SDMX (<http://sdmx.org>), or SCovo (<http://vocab.deri.ie/scovo>) are used. The VoID guidelines, for instance, recommend the use of SCovo to share statistical dataset features (Alexander et al. 2009). Auer et al. present LODStats (Auer et al. 2012), a framework

for dataset analytics, which introduces a set of 32 statistical features and uses the most recommended combination of VoID and the Data Cube vocabulary.

While there does exist a wealth of methods for assessing characteristics related to the *dynamic* and evolution of datasets, including the *Talis Changeset vocabulary* (<http://vocab.org/changeset/schema.html>), the Delta vocabulary (<http://www.w3.org/2004/delta>), RMO introduced by Graube et al. (2014), or the *Triplify Update vocabulary* (<http://triplify.org/vocabulary/update>), most vocabularies in this area are dedicated to representing the actual evolution of a dataset, rather than higher-level observations about dynamics. A more abstract approach is offered by the *Dataset Dynamics (DaDy) vocabulary* (<http://vocab.deri.ie/dady>), which allows the representation of more abstract dynamics-related observations for a specific dataset, to be used in conjunction with VoID.

Applications

This section illustrates the use of RDF dataset profiles in several cross-domain applications without being exhaustive.

Data linking applications aim to annotate, disambiguate, and interlink entities and events by often using natural language processing (NLP) techniques and external sources including Linked Data. In this context, popular services include DBpedia Spotlight (Daiber et al. 2013) and Babelfy (Moro et al. 2014). Data linking applications typically use features from the *General* category, such as *topics*, *domains*, or *representative elements*.

Applications for data curation, cleansing, and maintenance rely on or generate profile features in order to improve the overall data quality. This includes the application of statistical methods for outliers detection (correcting errors in numerical values) (e.g., in Paulheim and Bizer 2014), link correction, as well as error detection and correction by using existing links. Features from the *Statistical, Provenance and Dynamics*

categories are largely used in this group of applications. Note that new features can also be generated as an outcome of these applications.

Schema inference applications have been developed recently, aiming at filling the gap generated by the lack of explicit vocabularies or by incomplete specifications (e.g., Paulheim and Bizer 2014; Konrath et al. 2012). *Statistical* characteristics of datasets, together with *Provenance* features, play an important role for this type of applications.

Applications for query answering over distributed data comprise the generation of ordered query plans against the mediated schema on a number of data sources. In order to guide distributed query processing, existing applications rely on indices of varying granularity including schema-level indices and data summaries (e.g., Harth et al. 2010). The majority of existing query applications rely on the *General* and *Statistical* characteristics at the schema and data levels. Finally, quality-aware query applications also take into account features from the *Qualitative* category (e.g., *completeness* and *accuracy*).

Reuse of datasets can become apparent when datasets are linked to from other datasets. In this context, *Links* features can provide an indication of dataset reuse (Endris et al. 2017).

Future Challenges

Overall, applications of the whole spectrum of dataset profile feature categories can be found, including general, qualitative, statistical, and dynamic features discussed in this entry. However, individual applications as well as tools commonly use or support only a very limited number of features. Typical stakeholders of the discussed techniques are both data providers, such as archival organizations, libraries, or individual data hosters, and data consumers, for instance, application developers or (domain-specific) data retrieval and search engines. In addition, with respect to Web-scale dataset discovery, the heterogeneous and fragmented landscape of feature definitions and applied vocabularies hinders the automated interpretation of dataset profiles, requiring further effort for

consumers of dataset profiles for interpreting, mapping, and disambiguating existing profiles.

The growing adoption of schema.org markup, particularly of scientific, datasets, driven also by existing community efforts such as the W3C community group on *schema.org for Datasets* (<https://www.w3.org/community/schemaorg4datasets/>) has led to a widespread availability of semi-structured dataset annotations. However, while this constitutes an unprecedented source of dataset-centric information, the diverse and often poorly structured nature of embedded markup (Yu et al. 2017) poses the need for further processing of such dataset profiles, for instance, through disambiguating, resolving, and augmenting dataset descriptions.

While reconciliation of dataset profiles is a general issue of relevance beyond just embedded Web markup, future work has to provide a stronger emphasis on obtaining, resolving, consolidating, and cleaning datasets profiles from a wide variety of signals, including structured RDF dataset annotations, embedded Web page markup, but also less explicit indicators observable on the Web, for instance, in dataset registries, unstructured Web pages, or scientific publications.

Cross-References

- ▶ [Data Profiling](#)
- ▶ [Data Quality and Data Cleansing of Semantic Data](#)
- ▶ [Federated RDF Query Processing](#)
- ▶ [Semantic Interlinking](#)
- ▶ [Semantic Search](#)

References

- Abedjan Z, Grütze T, Jentzsch A, Naumann F (2014) Profiling and mining RDF data with prolod++. In: Proceedings of the 30th international conference on data engineering, ICDE 2014, Chicago, 31 Mar–4 Apr 2014, pp 1198–1201
- Alexander K, Cyganiak R, Hausenblas M, Zhao J (2009) Describing linked datasets – on the design and usage of void, the ‘vocabulary of interlinked datasets’. In: WWW 2009 workshop: linked data on the web (LDOW2009), Madrid

- Auer S, Demter J, Martin M, Lehmann J (2012) Lodstats – an extensible framework for high-performance dataset analytics. In: Proceedings of the 18th international conference on knowledge engineering and knowledge management, EKAW 2012, Galway City, 8–12 Oct 2012, pp 353–362
- Ben Ellef M, Bellahsene Z, John B, Demidova E, Dietze S, Szymanski J, Todorov K (2017) RDF dataset profiling – a survey of features, methods, vocabularies and applications. *Semant Web J*
- Bizer C, Cyganiak R (2009) Quality-driven information filtering using the WIQA policy framework. *J Web Sem* 7(1):1–10
- Böhm C, Lorey J, Naumann F (2011) Creating void descriptions for web-scale data. *J Web Sem* 9(3): 339–345
- Daiber J, Jakob M, Hokamp C, Mendes PN (2013) Improving efficiency and accuracy in multilingual entity extraction. In: Proceedings of the 9th international conference on semantic systems, I-SEMANTICS 2013, Graz, 4–6 Sept 2013, pp 121–124
- Debattista J, Lange C, Auer S (2014) daQ, an ontology for dataset Quality information. In: Proceedings of the workshop on linked data on the web co-located with the 23rd international world wide web conference (WWW 2014), Seoul, 8 Apr 2014
- Endris KM, Giménez-Garí JM, Thakkar H, Demidova E, Zimmermann A, Lange C, Simperl E (2017) Dataset reuse: an analysis of references in community discussions, publications and data. In: Proceedings of the ninth international conference on knowledge capture (K-CAP 2017)
- Fetahu B, Dietze S, Nunes BP, Casanova MA, Taibi D, Nejdl W (2014) A scalable approach for efficiently generating structured dataset topic profiles. In: Proceedings of the 11th ESWC conference 2014, Anissaras, 25–29 May 2014, pp 519–534
- Fürber C, Hepp M (2011) Towards a vocabulary for data quality management in semantic web architectures. In: Proceedings of the 1st international workshop on linked web data management, LWDM’11. ACM, New York, pp 1–8
- Graube M, Hensel S, Urbas L (2014) R43ples: revisions for triples – an approach for version control in the semantic web. In: Proceedings of the 1st workshop on linked data quality co-located with 10th international conference on semantic systems, LDQ@SEMANTiCS 2014, Leipzig, 2 Sept 2014
- Harth A, Hose K, Karnstedt M, Polleres A, Sattler KU, Umbrich J (2010) Data summaries for on-demand queries over linked data. In: Proceedings of the 19th international conference on world wide web, WWW’10. ACM, New York, pp 411–420
- Käfer T, Abdelrahman A, Umbrich J, O’Byrne P, Hogan A (2013) Observing linked data dynamics. In: Proceedings of the 10th ESWC conference, Montpellier, 26–30 May 2013, pp 213–227
- Konrath M, Gottron T, Staab S, Scherp A (2012) Schemex – efficient construction of a data catalogue by stream-based indexing of linked data. *J Web Sem* 16: 52–58
- Missier P, Belhajame K, Cheney J (2013) The W3C PROV family of specifications for modelling provenance metadata. In: Joint 2013 EDBT/ICDT conferences, EDBT’13. Proceedings, Genoa, 18–22 Mar 2013, pp 773–776
- Moro A, Raganato A, Navigli R (2014) Entity linking meets word sense disambiguation: a unified approach. *TACL* 2:231–244
- Omitola T, Zuo L, Gutteridge C, Millard IC, Glaser H, Gibbons N, Shadbolt N (2011) Tracing the provenance of linked data using void. In: Proceedings of the international conference on web intelligence, mining and semantics, WIMS’11. ACM, New York, pp 17:1–17:7
- Paulheim H, Bizer C (2014) Improving the quality of linked data using statistical distributions. *Int J Semant Web Inf Syst* 10(2):63–86
- Umbrich J, Neumaier S, Polleres A (2015) Quality assessment and evolution of open data portals. In: Proceedings of the 3rd international conference on future internet of things and cloud, FiCloud 2015, Rome, 24–26 Aug 2015, pp 404–411
- Yu R, Gadhiraju U, Fetahu B, Dietze S (2017) Fusem: query-centric data fusion on structured web markup. In: Proceedings of the 2017 IEEE 33nd international conference on data engineering (ICDE). IEEE
- Zaveri A, Rula A, Maurino A, Pietrobon R, Lehmann J, Auer S (2016) Quality assessment for linked data: a survey. *Semant Web* 7(1):63–93

RDF Formats

► [RDF Serialization and Archival](#)

RDF Graph Embeddings

► [Knowledge Graph Embeddings](#)

RDF Serialization and Archival

Javier D. Fernández¹ and Miguel A. Martínez-Prieto²

¹Complexity Science Hub Vienna, Vienna University of Economics and Business, Vienna, Austria

²Department of Computer Science, Universidad de Valladolid, Valladolid, Spain

Synonyms

[RDF formats](#); [RDF syntaxes](#)

Definitions

RDF serialization is the process of writing down RDF graphs into a machine-readable format. RDF formats mainly differ in the concrete syntax to serialize RDF statements (called “triples”) and how to group or nest a set of statements, influencing the amount of storage space and bandwidth required for preserving and exchanging such data. These differences can be rather marginal for small RDF graphs, where the selection of a particular format is mostly driven by user preferences, the set of tools managing the RDF format, and the interoperability with other applications. In contrast, choosing an adequate serialization format can affect the overall performance and present important scalability issues when managing Big Semantic Data collections.

Additional challenges arise in scenarios where triples must be annotated with information about their context, such as provenance, trust, or quality information, to name but a few. The most standard solution in RDF is to consider *named graphs*, i.e., different RDF graphs are managed under a single RDF dataset. Diverse RDF formats have been proposed to cover this scenario and serialize annotated statements (called “quads”), at the cost of paying additional costs to represent triples that can be repeated across graphs.

This situation is particularly challenging when different versions of an RDF graph must be preserved, given that graphs can be near-copies of others. This problem is commonly referred to as *RDF archival*, where specific archival policies have been proposed in recent literature.

This entry provides a historical review of RDF serialization formats to understand their evolution over the years. Basic features are covered for each format, paying special attention to its capabilities for quad serialization. XML-based and text-oriented formats are first introduced to illustrate how RDF was originally used for metadata description. Their limitations led to JSON-based syntaxes, which overcome some processing challenges, but do not scale to the high demanding needs of Big Semantic Data management. This fact motivates the proposal of binary formats,

able to deal with the storing and exchanging needs of large RDF collections. Finally, RDF archival proposals are surveyed and we conclude presenting open research trends.

Overview

RDF (*Resource Description Framework*) (Schreiber and Raimond 2014) proposes a logical model for expressing information about physical (e.g., people, buildings, vehicles, or pictures, among others) and abstract (e.g., films, songs, cities, etc.) resources. The information is represented as ternary relations, called RDF triples (or statements), which are organized into hyperlinked clouds, referred to as RDF graphs. The resulting knowledge is typically oriented for machine consumption. In the following, the main RDF concepts are briefly introduced.

RDF Triple. RDF statements are built on a simple (subject, predicate, object) structure called *RDF triple* (aka statement), which sets a particular value (the “object”) for a given feature (“predicate”) of the resource (“subject”) being described. For instance, an informal representation of a triple which sets the birth date of the singer Bruce Springsteen can be $\langle \text{Bruce Springsteen}, \text{is born on}, 1949-09-23 \rangle$.

The RDF data model (Cyganiak et al. 2014) establishes some restrictions about the universe of possible values for each component of a triple. Thus, a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$, where I stands for IRIs, B for blank nodes, and L for literals. These mutual disjoint sets of RDF terms are described as follows:

IRIs. International Resource Identifiers (IRIs) are used to identify resources in RDF. For instance, $\langle \text{http://example.org/Springsteen} \rangle$ is used to name the aforementioned resource about Bruce Springsteen. Note that IRIs can also be used to identify predicate and object values of a triple; e.g., $\langle \text{http://example.org/p/birthDate} \rangle$ is a valid IRI to identify the corresponding “birth date” property. IRIs are global identifiers, so they can be reused to provide additional information about a resource or to

convey the same meaning, e.g., a birth date feature in a different context.

Blank Nodes. RDF uses blank nodes (also called anonymous nodes) to declare the existence of a resource without using a particular IRI. Blank nodes can play both subject and object roles, while they never mean that the IRI is unknown for the corresponding resource. In turn, the scope of the blank node is limited to the RDF graph where it is used.

Literals. Final values (numbers, names, dates, etc.) are expressed as RDF literals, always used as objects. Literals are declared by default as strings (a language tag can optionally be associated in this case), but other datatypes can be used. The value “1949-09-23” is an example of an RDF literal.

Figure 1 illustrates the above RDF concepts. It comprises three triples that describe a new resource about Bruce Springsteen, identified by the aforementioned IRI <<http://example.org/Springsteen>>. Two of these triples set his birth name and birth date (using IRIs <<http://example.org/p/name>> and <<http://example.org/p/birthDate>>, respectively), i.e., “Bruce Frederick Joseph Springsteen” and “1949-09-23.” The third triple connects the new resource with an existing description of Bruce Springsteen in DBpedia (<http://dbpedia.org/page/Bruce_Springssteen>), a conversion of Wikipedia to RDF. Note that the predicate reuses the sameAs property, described in the OWL ontology.

RDF is traditionally modeled as a labeled directed graph (as seen in the previous figure) because it provides an easy-to-understand (visual) explanation of the RDF data. This fact motivates the adoption of this concept as part of the RDF model.

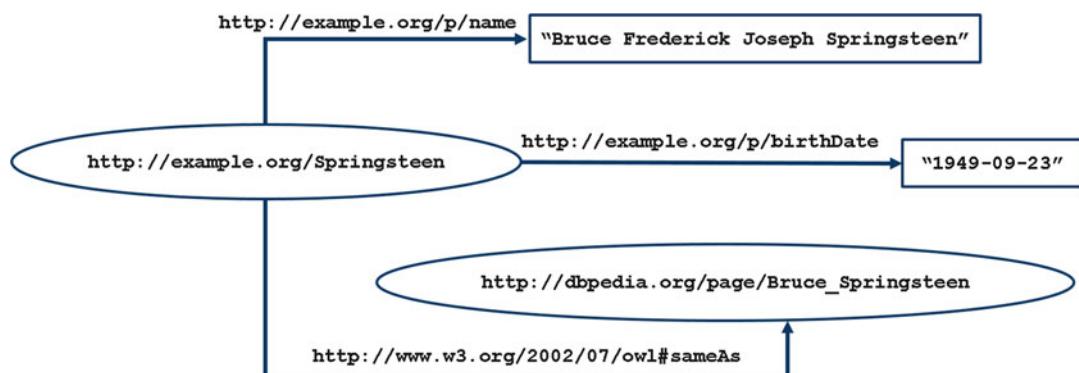
RDF Graph. An RDF graph G is a set of triples declared under the same scope. Thus, a triple belongs to an RDF graph, and a graph contains a well-determined set of triples.

It is worth noting that an RDF graph is only a “mental model,” and its triples must be serialized for preservation or exchanging purposes. Each serialization format has its particular features, but all ensure RDF graphs to be effectively written down.

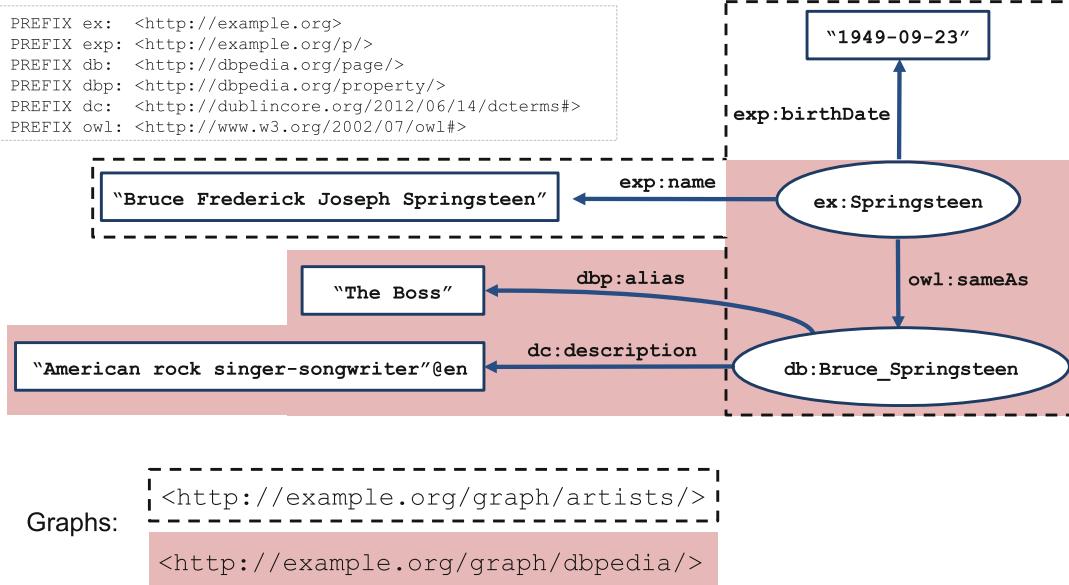
RDF 1.1 extends the original model to support grouping RDF graphs within a single RDF dataset, enabling triples from different contexts to be managed together.

RDF Dataset. An RDF dataset D is a collection of RDF graphs, where one of them is considered the “default graph.” The dataset contains zero or more “named graphs.” Each named graph is a pair consisting of an IRI or a blank node (the graph name) and an RDF graph. Graph names are unique within the RDF dataset, and blank nodes can be shared between graphs.

Figure 2 shows an RDF dataset which comprises two RDF graphs (note that a prefix notation is used to compact their IRIs). The first graph



RDF Serialization and Archival, Fig. 1 RDF graph which comprises three triples about *Bruce Springsteen*



RDF Serialization and Archival, Fig. 2 RDF dataset which comprises two RDF graphs

(dashed) is identified by the IRI <http://example.org/graph/artists> and includes the set of three triples showed in Fig. 1. The new named graph (solid background) declares three more triples about Bruce Springsteen, one of them shared with the original graph: (ex:Springsteen, owl:sameAs, db:Bruce_Springsteen).

The notion of RDF dataset also applies to a logical level. However, serializing triples from a dataset brings an additional requirement, as their context must be preserved.

RDF Quad. An RDF quad is an extended statement that includes the corresponding triple and the name of the graph that declares it (aka context). More formally, an RDF quad q is a quadruple $\langle \text{subject}, \text{predicate}, \text{object}, \text{graph} \rangle$, where graph refers to the name of a graph which exists in the dataset. Thus, a quad $(s, p, o, g) \in (I \cup B) \times I \times (U \cup B \cup L) \times (I \cup B)$.

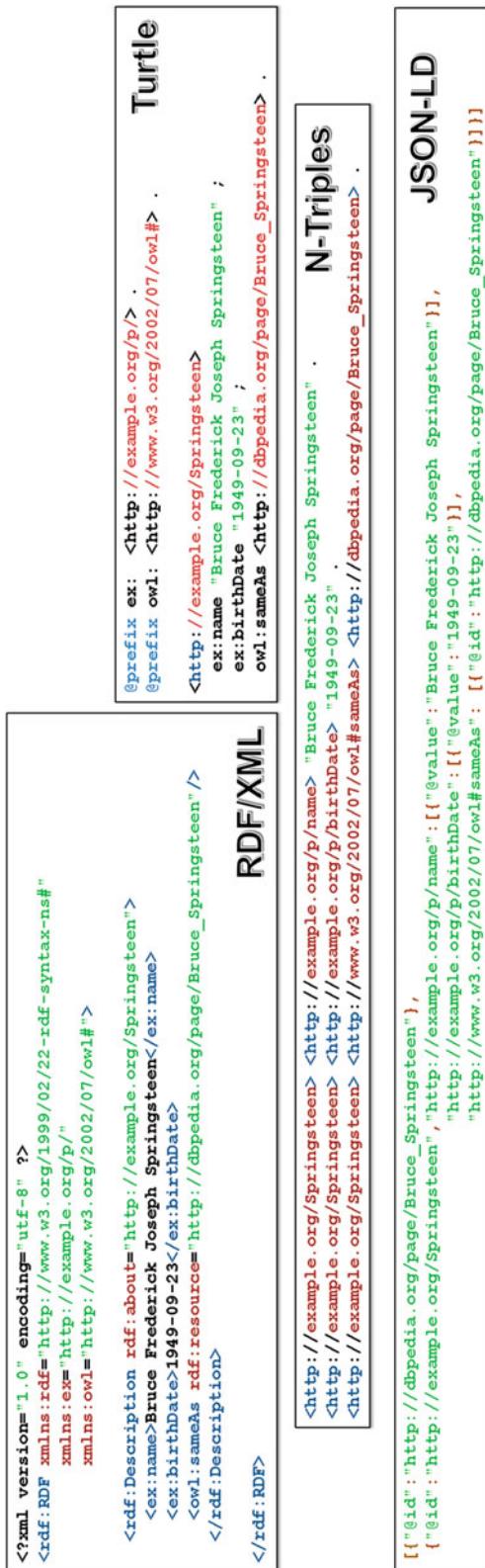
Serialization Formats

The RDF model describes the previous concepts using an abstract syntax, but it does not restrict how they are effectively serialized. Thus, RDF data can be written down in different ways, while

several serialization formats are standards and widely accepted by the Semantic Web community. These formats allow RDF graphs to be effectively serialized, but only some of them are able to cover particular RDF dataset needs.

RDF/XML (Gandon and Schreiber 2014) was released hand in hand with the initial W3C RDF Recommendation. In early dates, RDF/XML was meant to be an ideal first serialization for RDF graphs as it could leverage all XML-based solutions. However, RDF/XML overloads the representation with verbose human-focused information, which can serve the intended exchanging purposes, but only on a small scale. Nonetheless, RDF/XML includes some naive compacting features, such as the possibility to (i) implicitly create blank nodes without giving a concrete identifier, (ii) omit nodes and place values as property attributes in XML, (iii) abbreviate IRI references via base IRIs (namespaces) and relative references, and (iv) create collections to define a set of terms related to a subject.

Figure 3 shows an example of an RDF/XML serialization that encodes triples from Fig. 1. In practice, the result is an XML document, which can be parsed, processed, and queried using well-established technologies (DOM, XPath, XSLT,



RDF Serialization and Archival, Fig. 3 Serializations of triples from Fig. 1

etc.). However, its document orientation is an important weakness to deal with large amounts of RDF triples. Besides, it does not support named graphs.

Trix (Carroll and Stickler 2004) proposes another XML syntax for RDF which organizes triples by graphs, allowing multiple graphs to be serialized into the same document. It is a first approximation to an RDF dataset serialization, but the resulting format shows the same drawbacks that RDF/XML.

XML-based formats have lost relevance, and their usage is limited to small RDF graph serializations (e.g., descriptive metadata about a Web page).

N3 (Notation3) (Berners-Lee and Connolly 2011) is a format designed with human readability in mind. Although it may make sense in the first times of RDF, managing and processing Big Semantic Data are far from any human capability. However, this format breaks with the XML predominance and introduces some interesting constructors which tackle particular RDF features.

sN3 proposes the use of namespaces, as in XML. It is an effective compaction mechanism which allows relative IRIs to be declared to their corresponding namespace. On the other hand, N3 also introduces constructors for triples encoding in the form of adjacency lists: predicate lists allow subjects to be written only once for all triples containing it, while object lists concatenate all object values related with a pair `{subject, predicate}`, which is written once.

This format proposes some other constructors which goes beyond the needs of RDF serialization, making the format relatively complex for such purpose. N3 does not support quads.

N-Triples (Becket 2014) is an extremely simple line-based syntax, easy to parse and generate. In essence, the subject, predicate, and object terms are separated by a white space, and the triple is terminated with a “.” followed by a new line. IRIs are enclosed in “<” and “>” and literals in “ ”, and blank nodes start with “_:”. Figure 3 shows an N-Triples serialization that basically lists the corresponding triples. Note that

N-Triples writes down each full term as many times as it is used in a triple, resulting in a simple but extremely verbose serialization due to long-term repetitions. As a result, N-Triples files need much more space than others, which can result in scalability issues for Big Semantic Data management.

On the other hand, N-Triples can be easily extended to support quad serialization. It only needs the graph name to be appended to the triple. N-Quads (Carothers 2014) formalizes this approach, featuring the same characteristics and limitations as N-Triples.

Turtle (Beckett et al. 2014) is a widely used format that exploits the previous experience of N3 and N-Triples. On the one hand, it delimits the expressive power of N3 to only serialize valid RDF graphs. On the other hand, it addresses N-Triples drawbacks to consolidate a more practical format.

Figure 3 also shows a Turtle excerpt that illustrates some of its more relevant features. For instance, it shows the use of namespaces. Note that each one is declared by the `@prefix` constructor, while IRIs in the terms are rewritten in relative form to their corresponding namespaces. The figure also illustrates the predicate list encoding proposed in N3; e.g., `http://example.org/Springsteen` is written once, but it plays the role of a subject for three different triples. Turtle supports object lists too, and it introduces more constructors and different kinds of syntactic sugar to alleviate RDF verbosity.

Although Turtle is a popular format, it does not support quads. As in the previous case, a new format, called TriG (Bizer and Cyganiak 2014), extends Turtle to allow RDF dataset serialization. It basically encloses triples that belong to each named graph in the dataset.

JSON-LD (Sporny et al. 2014) exploits JSON features to serialize RDF. It comes with the advantage of using a well-established scheme that is easy to parse and widely accepted by Web APIs. The main focus, then, is to be easy for humans to read and write and easy for machines to parse and generate automatically. JSON-LD is designed to be usable directly as JSON, with no knowledge of RDF. Note that JSON-LD supports named graphs

natively, and it is gaining increasingly attention by the community.

Key Research Findings

The above serialization formats have been successfully used for managing small- and medium-sized RDF graphs. However, the steady adoption of RDF, in particular in the context of linked data (Bizer et al. 2009), brings larger graphs including hundreds of millions and even billion triples. For instance, the latest version of DBpedia (2016–10), an RDF conversion of Wikipedia, consists of roughly 13 billion triples, and LOD Laundromat (Beek et al. 2014), a service crawling RDF datasets, reports that around 4000 datasets contain more than 1 million triples.

In addition, named graphs are increasingly incorporated to consolidate complex RDF datasets. However, formats for quads are less mature and also suffer from the lack of scalability. This problem is particularly challenging when the corresponding RDF dataset is a historical archive of a graph, containing its different states over the time.

This section delves into detail of the most innovative binary serialization formats, designed with volume issues in mind in order to solve the aforementioned scalability issues. Some of them also cover quad management, although managing context information is a challenge by itself, which is also reviewed below. Finally, RDF archival foundations are introduced, summarizing the most recent approaches.

Binary Serialization and Compression

Traditional RDF formats were not designed for a scenario of large-scale and machine-understandable Web of data. Their syntaxes have constructors which organizes RDF statements in a human-readable way that adds unnecessary overheads for storing, exchanging, and consuming RDF graphs. Although this scalability issue can be partially solved through universal compression (e.g., gzip or bzip2) over such formats, specific RDF binary serializations and compressors have been also proposed.

These tailored solutions mostly focus on taking advantage of particular features of RDF data in order to reduce the verbosity and produce important space savings at large scale. In the following, the three most prominent binary serializations are briefly reviewed: *HDT*, *RDF Binary*, and *RDF4J*. We then list solutions focused on streaming and provide a summary of RDF compression techniques to provide a big picture of the current state of the art (the interested reader can find a chapter specifically devoted to RDF compression).

The HDT (Fernández et al. 2011, 2013) format proposes a binary syntax for RDF data focused on producing very compact serializations to speed up data exchange, but also efficient data parsing and access. HDT minimizes the repetition of terms (IRIs, blank nodes, and literals) using the so-called HDT Dictionary, which assigns a numerical ID to each different term. Then, the graph structure of the dataset is managed as a graph of (term) IDs, in the HDT Triples component. Both dictionary and triples components are then compacted (e.g., looking for common string prefixes in the terms) and partially indexed. HDT is one of the most widespread RDF binary formats, mainly due to the HDT adoption as a compact data store for LOD Laundromat (Beek et al. 2014), and the data back end of lightweight APIs such as Triple Pattern Fragments (Verborgh et al. 2016).

HDT traditionally focuses on representing single RDF graphs. A recent approach, named HDTQ (Fernández et al. 2018), extends HDT to represent named graphs, keeping compact and retrieval features.

The RDF binary format (RDF Binary 2017) is an alternative solution proposed by the well-known Jena semantic framework. It consists of very simple mappings to encode triples in Apache Thrift (Apache Thrift 2017), which provides a scalable cross-language platform. In this case, rather than compactness, RDF binary mostly focuses on avoiding to parse the textual RDF triples; hence, the overall processing is sped up. RDF binary supports both RDF graphs and RDF datasets (named graphs) encoded as a stream of quads.

The RDF4j (RDF4j 2017) binary format is proposed and used within the Eclipse RDF4J framework. The RDF4j format partially combines both previous strategies. On the one hand, it mostly tackles parsing and processing efficiency, providing a concrete syntax to delimit the extent of each term and triple. On the other hand, it allows for an in-line declaration of a dictionary, where a term is mapped to an ID which can be referred in another triple. Nonetheless, terms are not compressed themselves (e.g., using prefixes such as in HDT); hence, only partial compression is achieved.

Compression is another way of serializing RDF. As explained, combining universal compression and any serialization format is a common practice, but different compressors have been designed from the scratch to deal with particular RDF requirements. RDF compressors can be classified into *physical* and *logical* compressors. Physical compressors (Fernández et al. 2013; Swacha and Grabowski 2015; Álvarez-García et al. 2014; Brisaboa et al. 2015) exploit symbolic/syntactic redundancy, removing term repetitions and compacting repetitive subgraph structures underlying to the dataset. In contrast, logical compressors (Iannone et al. 2005; Meier 2008; Joshi et al. 2013; Venkataraman and Sreenivasa Kumar 2015) focus on semantic-based redundancy, avoiding to represent triples that can be inferred from others in the RDF graph.

In addition, diverse binary formats and compressors have been proposed for RDF streams, i.e., a continuous flow of RDF data. In this case, the challenge consists of exploiting the trade-offs between the space savings achieved by the format and the latency introduced in the creation and parsing processes. Streaming HDT (Hasemann et al. 2012) adapts HDT to simplify the process by restricting the carried metadata and the maximum length of the dictionary; hence, shorter IDs are used. RDSZ (Fernández et al. 2014b) uses differential encoding to compact the similarities between consecutive triples in the stream. ERI (Fernández et al. 2014a) is an RDF stream compressor that adapts the W3C Efficient XML Interchange (EXI) format (Schneider et al.

2014) for RDF data. Note that EXI encoding can also be directly applied over an RDF/XML or JSON serialization. PatBin (Lhez et al. 2017) and FSSD (Karim et al. 2017) perform dictionary-based compression together with pattern-based encoding.

Context Information

As stated, graph names are increasingly used to capture additional information such as trust, provenance, temporal information and other annotations (Carroll et al. 2005; Zimmermann et al. 2012). Although there exist standard RDF syntaxes (such as N-Quads, Trig or JSON-LD) that represent RDF named graphs, serializing annotated RDF data (quads) efficiently remains an open challenge.

In spite of general approaches, such as AnQL (Zimmermann et al. 2012), most solutions focus on managing provenance information, as this is at the core of the linked data distributed philosophy (Bizer et al. 2009). Besides the aforementioned named graphs and the standard RDF reification (Schreiber and Raimond 2014), i.e., using the RDF vocabulary (*rdf:Statement*, *rdf:subject*, *rdf:predicate*, and *rdf:object*) to refer to statements, the main proposals are singleton properties (Nguyen et al. 2014) and N-ary relations (Noy et al. 2006). The former introduces unique predicates that are then annotated with the metadata of the triple it belongs to. The latest, used in Wikidata, represents a relation between a subject and object with a new resource, which is then connected to the subject, on the one hand, and predicate and object, on the other. Further information can be attached to the new resource in order to annotate the statement.

In addition, two recent solutions have been proposed. (Hartig 2017) extends RDF with a notion of embedded triples (encoded between ‘<<’ and ‘>>’), which can be directly used as subject or object of other triples. NdFluents (Giménez-García et al. 2017) creates unique versions of the subject and the object for each annotated triple, which are then linked to a context resource and to the original subject and object resources.

RDF Archival

RDF archival is a particular instance of the problem of managing context information. In this case, the context is set by the moment when a new version of an RDF graph is released. In general, RDF data are not static but evolve naturally, without centralized monitoring nor further advise, following the scale-free nature of the Web. Thus, RDF archiving emerges as a novel challenge aimed at assuring quality and traceability of RDF data over time.

On a high level, the World Wide Web Consortium (W3C) provides basic guidelines on how to perform data versioning on datasets published in the Web (Lóscio et al. 2017). The set of recommendations includes (i) providing a version indicator (e.g., via *owl:versionInfo*); (ii) serving different versions via the Memento framework (de Sompel et al. 2010), which can provide access to prior states of RDF resources using date-time negotiation in HTTP; and (iii) providing the changes made in each version. Nonetheless, these recommendations are generic and do not restrict how RDF data versions are stored or queried across time. Initial works on RDF archiving policies and systems are starting to address these issues, proposing different solutions to efficiently archive and query different versions of RDF data.

Main efforts on RDF archiving fall in one of the following four storage strategies: *independent copies (IC)* and *change-based (CB)* and *timestamp-based (TB)* and *hybrid-based (HB)* approaches.

Independent copies (IC) (Klein et al. 2002; Noy and Musen 2004) is the most naive approach where each version (aka snapshot) is managed as a different, complete graph. On the one hand, IC faces scalability problems as static information is duplicated across the versions. In addition, some operations such as knowing the difference between versions require non-negligible processing efforts. On the other hand, version materialization (retrieve certain version) is as efficient as querying a single snapshot.

Change-based approach (CB) (Volkel et al. 2005; Dong-Hyuk et al. 2012; Zeginis et al. 2011) partially addresses the space issues of IC by storing the differences (deltas) between versions.

In contrast, CB requires additional computational costs for retrieving a particular version given that deltas need to be propagated.

Timestamp-based approach (TB) (Cerdeira-Pena et al. 2016; Gutierrez et al. 2007; Zimmermann et al. 2012) annotates each triple with its temporal validity, i.e., the version. Compression techniques can be used to minimize the space overheads, e.g., using self-indexes, such as in v-RDFCSA (Cerdeira-Pena et al. 2016), or delta compression in B+Trees (Zaniolo 2016).

Hybrid-based approaches (HB) (Stefanidis et al. 2014; Neumann and Weikum 2010; Zaniolo 2016) combine previous policies to inspect other space/performance trade-offs. In particular, the hybrid IC/CB approach (Dong-Hyuk et al. 2012; Meinhardt et al. 2015; Stefanidis et al. 2014) follows a CB solution where full version materialization is additionally provided in some intermediate steps; hence, delta propagation is mitigated. In contrast, other practical approaches (Graube et al. 2014; Neumann and Weikum 2010; Vander Sander et al. 2013; Zaniolo 2016) follow a TB/CB approach in which triples can be time-annotated only when they are added or deleted. Although this reduces the space needs (as it manages less annotations), version materialization requires to rebuild the delta similarly to CB.

R

Future Directions for Research

As a result of standardization efforts by the Semantic Web community, there are many diverse standard “plain” RDF serializations available. Despite potential future trends that may result in adaptations for RDF (such as JSON-LD, adapted from JSON), most research efforts focus on efficient representation of annotated triples, in particular to model provenance information (Giménez-García et al. 2017; Hartig 2017).

RDF binary formats and compression have also emerged as active research and development fields over the past years. The main reason is that (i) current plain RDF formats are dominated by a human-centric view and suffer from scal-

ability problems at large scale and (ii) general compressed solutions still miss some types of redundancy underlying to RDF data. In this regard, there is still room for hybrid compressors leveraging syntactic and semantic redundancies. Then, RDF self-indexing (i.e., compressed and indexed RDF data) is still a main direction for research, in particular in the unexplored field of RDF streaming.

Finally, the community is just starting to face serious scalability issues for RDF archival. In the absence of a scalable archival approach at Web scale, RDF data change and vanish without further notice nor trace of previous versions. Future directions in this regard include further research on scalable archival methods (potentially distributed) as well as efficient mechanisms to resolve structured cross-time queries.

Cross-References

► RDF Compression

References

- Álvarez-García S, Brisaboa N, Fernández JD, Martínez-Prieto MA, Navarro G (2014) Compressed vertical partitioning for efficient RDF management. *Knowl Inf Syst* 44(2):439–474
- Apache Thrift (2017) Apache thrift. <https://thrift.apache.org/>
- Beckett D (2014) RDF 1.1 N-Triples: a line-based syntax for an RDF graph. W3C recommendation. <https://www.w3.org/TR/n-triples/>
- Beckett D, Berners-Lee T, Prud'hommeaux E, Carothers G (2014) RDF 1.1 turtle: terse RDF triple language. W3C recommendation. <https://www.w3.org/TR/turtle/>
- Beek W, Rietveld L, Bazoobandi HR, Wielemaker J, Schlobach S (2014) LOD laundromat: a uniform way of publishing other people's dirty data. In: 13th international semantic web conference (ISWC), pp 213–228
- Berners-Lee B, Connolly D (2011) Notation3 (N3): a readable RDF syntax. W3C team submission. <https://www.w3.org/TeamSubmission/n3/>
- Bizer C, Cyganiak R (2014) RDF 1.1 TriG: RDF dataset language. W3C recommendation. <https://www.w3.org/TR/trig/>
- Bizer C, Heath T, Berners-Lee T (2009) Linked data—the story so far. *Int J Semant Web Inf Syst* 5(3):1–22
- Brisaboa N, Cerdeira-Pena A, Farina A, Navarro G (2015) A compact RDF store using suffix arrays. In: 22nd international symposium on string processing and information retrieval (SPIRE), pp 103–115
- Carothers G (2014) RDF 1.1 N-Quads: A Line-based syntax for an RDF dataset. W3C recommendation. <https://www.w3.org/TR/n-quads/>
- Carroll J, Stickler P (2004) TriX : RDF triples in XML. Technical report, Digital Media Systems Laboratory, HP Laboratories Bristol
- Carroll JJ, Bizer C, Hayes P, Stickler P (2005) Named graphs, provenance and trust. In: Proceedings of the 14th international conference on World Wide Web. ACM, pp 613–622
- Cerdeira-Pena A, Farina A, Fernández JD, Martínez-Prieto MA (2016) Self-indexing RDF archives. In: Proceeding of DCC
- Cyganiak R, Wood D, Lanthaler M (2014) RDF 1.1 concepts and abstract syntax. W3C recommendation. <http://www.w3.org/TR/2014/REC-rdf11-abstract-20140225/>
- de Sompel HV, Sanderson R, Nelson ML, Balakireva L, Shankar H, Ainsworth S (2010) An HTTP-based versioning mechanism for linked data. In: Proceeding of LDOW
- Dong-Hyuk I, Sang-Won L, Hyoung-Joo K (2012) A version management framework for RDF triple stores. *Int J Softw Eng Knowl* 22(1):85–106
- Fernández JD, Martínez-Prieto MA, Gutiérrez C, Polleres A (2011) Binary RDF representation for publication and exchange (HDT). W3C member submission. <http://www.w3.org/Submission/HDT/>
- Fernández JD, Martínez-Prieto MA, Gutiérrez C, Polleres A, Arias M (2013) Binary RDF representation for publication and exchange. *J Web Semant* 19:22–41
- Fernández JD, Llaves A, Corcho O (2014a) Efficient RDF interchange (ERI) format for RDF data streams. In: 13th international semantic web conference (ISWC), pp 244–259
- Fernández N, Arias J, Sánchez L, Fuentes-Lorenzo D, Corcho Ó (2014b) RDSZ: an approach for lossless RDF stream compression. In: 11th European conference on the semantic web (ESWC), pp 52–67
- Fernández JD, Martínez-Prieto MA, Polleres A, Reindorf J (2018) HDTQ: managing RDF datasets in compressed space. In: European semantic web conference
- Gandon F, Schreiber G (2014) RDF 1.1 XML syntax. W3C recommendation. <https://www.w3.org/TR/rdf-syntax-grammar/>
- Giménez-García JM, Zimmermann A, Maret P (2017) Ndfluent: an ontology for annotated statements with inference preservation. In: European semantic web conference. Springer, pp 638–654
- Graube M, Hensel S, Urbas L (2014) R43ples: revisions for triples. In: Proceeding of LDQ, vol CEUR-WS 1215, paper 3
- Gutierrez C, Hurtado C, Vaisman A (2007) Introducing time into RDF. *IEEE Trans Knowl Data Eng* 19(2):207–218
- Hartig O (2017) Foundations of RDF* and SPARQL* – an alternative approach to statement-level metadata in RDF. In: Proceeding of AMW

- Hasemann H, Krolle A, Pagel M (2012) RDF provisioning for the internet of things. In: 3rd international conference on the internet of things (IOT), pp 143–150
- Iannone L, Palmisano I, Redavid D (2005) Optimizing RDF storage removing redundancies: an algorithm. In: 18th international conference on industrial and engineering applications of artificial intelligence and expert systems (IEA/AIE), pp 732–742
- Joshi A, Hitzler P, Dong G (2013) Logical linked data compression. In: 10th extended semantic Web conference (ESWC), pp 170–184
- Karim F, Vidal ME, Auer S (2017) Efficient processing of semantically represented sensor data. In: 13th international conference on Web information systems and technologies (WEBIST), pp 252–259
- Klein M, Fensel D, Kiryakov A, Ognyanov D (2002) Ontology versioning and change detection on the Web. In: Proceeding of EKAW, pp 197–212
- Lhez J, Ren X, Belabbess B, Curé O (2017) A compressed, inference-enabled encoding scheme for RDF stream processing. In: 14th European conference on the semantic Web (ESWC), pp 79–93
- Lóscio BF, Burle C, Calegari N (2017) Data on the web best practices. W3C recommendation 31 Jan 2017
- Meier M (2008) Towards rule-based minimization of RDF graphs under constraints. In: 2nd international conference on web reasoning and rule systems (RR), pp 89–103
- Meinhardt P, Knuth M, Sack H (2015) Tailr: a platform for preserving history on the web of data. In: Proceeding of SEMANTiCS. ACM, pp 57–64
- Neumann T, Weikum G (2010) x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proc VLDB Endow 3(1–2):256–263
- Nguyen V, Bodenreider O, Sheth A (2014) Don't like RDF reification? Making statements about statements using singleton property. In: Proceedings of the 23rd international conference on World Wide Web. ACM, pp 759–770
- Noy NF, Musen MA (2004) Ontology versioning in an ontology management framework. IEEE Intell Syst 19(4):6–13. <https://doi.org/10.1109/MIS.2004.33>
- Noy N, Rector A, Hayes P, Welty C (2006) Defining n-ary relations on the semantic web. W3C working group note 12(4)
- RDF Binary (2017) RDF binary using apache thrift. <https://jena.apache.org/documentation/io/rdf-binary.html>
- RDF4j (2017) Rdf4j binary RDF format. <http://docs.rdf4j.org/rdf4j-binary/>
- Schneider J, Kamiya T, Peintner D, Kyusakov R (2014) Efficient XML interchange (EXI) Format 1.0. W3C recommendation
- Schreiber G, Raimond Y (2014) RDF 1.1 primer. W3C working group note. <https://www.w3.org/TR/rdf11-primer/>
- Sporry M, Longley D, Kellogg G, Lanthaler M, Lindström N (2014) JSON-LD 1.0: a JSON-based serial-
ization for linked data. W3C recommendation. <https://www.w3.org/TR/json-ld/>
- Stefanidis K, Chrysakis I, Flouris G (2014) On designing archiving policies for evolving RDF datasets on the Web. In: Proceeding of ER, pp 43–56
- Swacha J, Grabowski S (2015) OFR: an efficient representation of RDF datasets. In: 4th symposium on languages, applications and technologies (SLATE), pp 224–235
- Vander Sander M, Colpaert P, Verborgh R, Coppens S, Mannens E, Van de Walle R (2013) R&Wbase: git for triples. In: Proceeding of LDOW
- Venkataraman G, Sreenivasa Kumar P (2015) Horn-rule based compression technique for RDF data. In: 30th annual ACM symposium on applied computing (SAC), pp 396–401
- Verborgh R, Vander Sande M, Hartig O, Van Herwegen J, De Vocht L, De Meester B, Haesendonck G, Colpaert P (2016) Triple pattern fragments: a low-cost knowledge graph interface for the Web. J Web Semant 37–38: 184–206
- Volkel M, Winkler W, Sure Y, Kruk S, Synak M (2005) Semversion: a versioning system for RDF and ontologies. In: Proceeding of ESWC
- Zaniolo SGJGC (2016) RDF-TX: a fast, user-friendly system for querying the history of RDF knowledge bases. In: Proceeding of EDBT
- Zeginis D, Tzitzikas Y, Christophides V (2011) On computing deltas of RDF/S knowledge bases. ACM Trans Web (TWEB) 5(3):14
- Zimmermann A, Lopes N, Polleres A, Straccia U (2012) A general framework for representing, reasoning and querying with annotated semantic Web data. JWS 12:72–95

R

RDF Stream Processing

► Semantic Stream Processing

RDF Syntaxes

► RDF Serialization and Archival

Real-Time Big Spatial Data Processing

► Streaming Big Spatial Data

Real-Time Streaming Benchmarks

► [Analytics Benchmarks](#)

Reasoning at Scale

Jacopo Urbani

Vrije Universiteit Amsterdam, Amsterdam,
The Netherlands

Synonyms

[Inference](#); [Knowledge base](#); [Rule-Based processing](#)

Definitions

Reasoning is the process of deriving new conclusions from knowledge bases using a series of logical steps. Reasoning at scale refers to the ability of applying this process to very large knowledge bases, such as modern knowledge graphs that are available on the Web.

Overview

The Web contains a very large amount of semi-structured datasets that cover encyclopedic knowledge, social or co-authorship networks, experimental results, etc. This data is encoded using RDF (Brickley et al. 2014), and it is interlinked to each other following the principles of linked open data, thus forming a large network of datasets called the Web of Data (WoD) (Bizer et al. 2009).

Automated reasoning can derive a wealth of nontrivial knowledge from these datasets, which can be used, for instance, to augment the WoD with new knowledge or to detect inconsistencies. Unfortunately, the large size of the WoD makes reasoning a challenging task. In fact, the datasets

might be too large to be stored in a single machine, requiring thus some form of distributed computing. Moreover, parallelizing the computation is not trivial due to factors like the input skewness or special corner cases that require sequential processing.

What constitutes the state of the art for reasoning on a large scale? To answer this question, this chapter offers a broad overview of the most recent efforts to execute rule-based reasoning on large inputs. More in particular, it describes the most important optimizations that can be applied to improve the efficiency of reasoning. Some of these optimizations work only with specific rules, while others are more generic. Even though none of them work with all possible inputs, in practice they turned out to be very effective as they enabled reasoning on large knowledge bases, with up to 100 billion triples in the largest experiments.

What Is Scale?

Generally speaking, the term scalability refers to the ability of a system to handle larger instances of a given problem. There can be several reasons that hinder the scalability of a system. First, the problem might have an unfavorable computational complexity which precludes termination within a reasonable time. Second, the algorithms might be poorly implemented. In this case, the system cannot scale well despite the problem is tractable. Third, the hardware might not have enough resources to carry on the computation. Scalability is thus a property that can be judged from three different angles: the theoretical complexity, the implementation, and the hardware requirements.

Before discussing the state of the art and describe how it deals with these challenges, it is important to define more precisely what reasoning is supposed to compute. Let KB be a generic RDF dataset, that is, a set of RDF statements. This dataset can be represented as a labeled directed graph where each triple $\langle s, p, o \rangle$ maps to an edge that connects s to o and is labeled with p . These graphs are typically called *knowledge graphs*. Let

$KG = (V, E)$ be such a knowledge graph where V is the set of entities and E the labeled edges that connect the entities. Given in input a knowledge graph KG , the goal of reasoning is to derive new knowledge that can be inferred from KG . This knowledge takes the form of new triples which can be logically deduced from the KG. For now, it is assumed that V is complete, i.e., KG already contains all the entities of interest. With this assumption in mind, then the triples derived from reasoning can be represented by new edges in KG .

The derivation of new triples is determined by a set of rules, which must be provided as input. Rules are expressions of the form

$$B_1, \dots, B_n \rightarrow H \quad (1)$$

where B_1, \dots, B_n are called *atoms*. An *atom* is an expression $p(\mathbf{x})$ where p is a *predicate* and $\mathbf{x} = x_1, \dots, x_m$ is a tuple of terms that can be either variables or constants. A *fact* is an atom without any variable. In our context, facts are used to represent the KG. They can be unary (e.g., to express the *isA* relation – $Person(Mark)$), binary (e.g., $livesIn(Mark, Amsterdam)$), or ternary (e.g., $T(Mark, livesIn, Amsterdam)$). The set of atoms B_1, \dots, B_n is called the rule's *body*, while H is the rule's *head*.

Notice that rules can be more complex than in (1). For instance, some body atoms might be negated, or the head might contain a conjunction of multiple atoms. All scalable approaches which will be discussed in this chapter assume that rules only contain positive atoms and only one atom occurs in the head of the rule. Moreover, they assume that every variable in the head must also appear in the body (safeness condition). The reason behind these constraints is that they simplify the computation. For instance, negation can introduce non-determinism, while dropping safeness might lead to nontermination.

The computation of the rules can be formalized as follows. Let I be a generic database of facts (i.e., the input KG); σ be a *substitution*, i.e., a partial mapping from variables to other variables or constants; and $r \in P$ be a rule of the

form (1) in the program P . Then, $r(I) = \{H\sigma \mid B_1\sigma, \dots, B_n\sigma \in I\}$ is the set of derivations that can be derived from I using r and $P(I) = \bigcup_{r \in P} r(I)$ is its extension to all rules in the program. The exhaustive application of all rules can be defined recursively by setting $P^0(I) = I$ and $P^{i+1}(I) = P^i(I) \cup P(P^i(I))$. Since the rules are safe and the set of constants is finite, there will be a j s.t. $P^{j+1}(I) = P^j(I)$. In this case, $P^j(I)$ is called the *closure* or *materialization* of I with P .

Materialization with Fixed Rules

Ontological languages are used to serialize semantic relations in RDF knowledge bases in a machine-readable format. For instance, they allow the user to define various semantic relations like subsumption between classes (e.g., *Student* is a subclass of *Person*) or specify that a relation is transitive (e.g., *ancestorOf* or *partOf*).

Ontological statements like the previous two examples can be translated into rules by either considering the standard constructs of the language or by also including the ontology at hand. The following example is useful to understand this difference.

Example 1 Let us assume that the KG contains the following ontological statements: $(:Bob, \text{isA}, :Actor)$ and $(:Actor, \text{soc}, :Man)$ where *soc* is an abbreviation for the standard RDF schema IRI of class subsumption.

A rule of the first type could be

$$\begin{aligned} T(A, \text{isA}, B), T(B, \text{soc}, C) \\ \rightarrow T(A, \text{isA}, C) \end{aligned} \quad (2)$$

while a rule of the second type could be

$$isA(A, :Actor) \rightarrow isA(A, :Man) \quad (3)$$

In the first case, rule (2) simply translates the inference that it is possible to obtain considering the *isA* and *soc* relations. The rule is domain-independent and can be applied to any KG. In

contrast, rule (3) is simpler because it does not require any data join but it has the disadvantage that it can be applied only to the input dataset.

In this chapter, rules of the first type are called *standard rules* since they are derived by standard ontological languages like RDF schema (Brickley et al. 2014) or OWL (Motik et al. 2009). Standard rules are important because they are universal in the sense that they do not depend on a particular input. Therefore, it is possible to introduce tailor-made optimizations to speed up their execution without any loss of generality. However, there are cases when nonstandard rules are preferable since they might be easier to execute. The remaining of this section will describe five optimizations which are crucial to enhance the scalability of reasoning using standard rules. The next section will address scalability with nonstandard rules.

Split instance/schema triples. The first, and perhaps most effective, optimization on current knowledge bases consists of splitting the input statements between the ones that describe instances and the ones that describe the schema. The last type of statements is typically ontological statements that use constructs from the language (e.g., OWL). One key property of current large KGs is that they contain many more instance statements than schema ones, and this is important because there are many standard rules which have two body atoms, one which matches instance statements while the other matches schema ones (Urbani et al. 2012). Rule (2) is such an example: Here, typically there will be many more triples of the form $T(A, \text{isA}, B)$ than of the form $T(B, \text{soc}, C)$.

A strategy to parallelize the computation of such rules is to simply range-partition the instance triples and assign each range to a different processor. If the processors operate in separated memory spaces (e.g., different machines), then schema triples can be replicated on each space.

After the partitioning is done, the rule can be executed in parallel without any intermediate node communication. An example of such computation is graphically depicted in Fig. 1a.

Reducing duplicates. There are cases where different rules might produce the same derivation. For instance, the two standard rules

$$\begin{aligned} T(A, P, B), T(P, \text{domain}, C) \\ \rightarrow T(A, \text{isA}, C) \end{aligned} \quad (4)$$

$$\begin{aligned} T(B, P, A), T(P, \text{range}, C) \\ \rightarrow T(A, \text{isA}, C) \end{aligned} \quad (5)$$

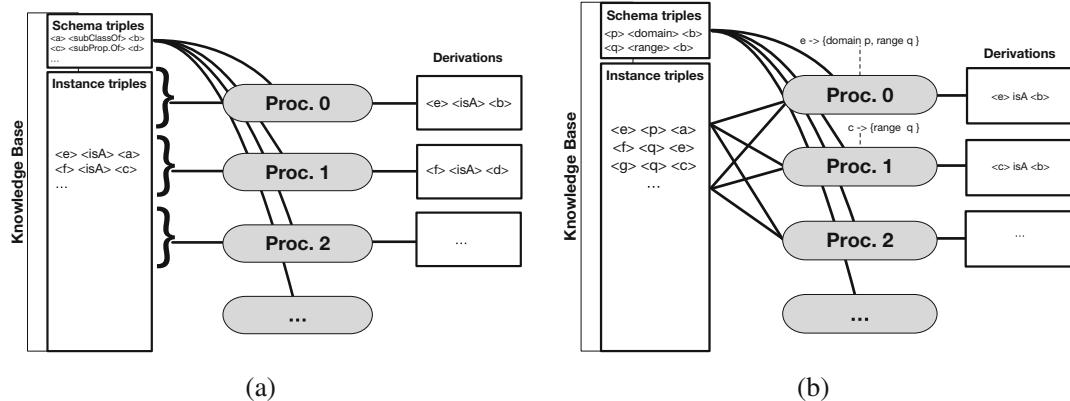
can derive the same information for an entity which is used both as a subject and as an object in different triples. For instance, $\langle Bob, \text{worksIn}, \text{Amsterdam} \rangle$ and $\langle Alice, \text{daughterOf}, Bob \rangle$ can both lead to the derivation $\langle Bob, \text{isA}, \text{Person} \rangle$ if the domain and range of the two predicates are *Person*.

A strategy to remove these types of duplicates is to group instance triples by the list of terms which are used in the head. In the previous case, triples can be grouped either by subject or by object, depending on the rule. Then, rules can be executed in parallel on each group. In this way, it is impossible that two different groups will produce the same derivation because the grouping criterion ensures that derivations must differ by at least one term (i.e., the grouping key).

During the rule computation, duplicates can still be produced within the same group. However, in this case they can be removed in parallel without any synchronization between the various processors. Figure 1b shows an example of such computation for rules (4) and (5).

sameAs table. One category of rules which is widely used in modern knowledge bases encodes reasoning over the equality of concepts, which is a relation that is stated with the predicate *owl:sameAs*. Equality is transitive (if a is the same as b and b is the same as c , then a is the same as c) and symmetric (if a is the same as b , then b is the same as a); thus rules in this category produce a large number of materializations.

To improve the performance, reasoners typically avoid materializing all conclusions but instead build a dedicated table where groups of



Reasoning at Scale, Fig. 1 (a) Parallel execution of a rule that requires a join between schema and instance triples. (b) Execution of a group of rules: in this example,

processor 0 receives all information regarding the entity “e,” which allows it to apply both rules (4) and (5) and remove duplicates locally

equal terms receive a unique representative ID, and all occurrences of these terms in the KB are replaced by the corresponding ID. During the materialization, the reasoner might derive more equality relations. If this happens, then the table needs to be updated, and more occurrences must be replaced with the corresponding ID. Notice that also rules need to be rewritten if they contain constants in the table.

After the materialization is computed, the augmented $P^j(I)$ will contain a compressed version of the full materialization since further derivations can be obtained by simply replacing the occurrences of representative IDs with each of the members of their equality group. However, in practice this operation is often omitted as it can be trivially computed on-the-fly whenever is needed.

Sorting rule execution. For some standard rule sets like the rules from RDF schema, it is possible to define a rule application order to reduce to the minimum the chance that the output of one rule can be used as input for another one. Unfortunately, this optimization guarantees a complete output only on some specific cases, and it is not possible to define an execution order which avoids repeated executions with more complex rulesets like, for instance, the OWL2 RL/RDF ruleset (Motik et al. 2009).

Memoization. Another technique that can be applied to improve the performance is *memoization* (Urbani et al. 2014). Memoization is a special type of caching which consists of storing the output of expensive functions to reduce the cost of repeated executions. For the problem of reasoning, memoization can be used to precompute all answers of some particular atoms. This enables a faster computation of data joins.

An example is useful to clarify this optimization. Let us consider, once again, rule (2). This rule contains two body atoms: One atom matches instance triples while the other matches schema triples. During the materialization, the number of facts that match the second atom will change if other rules derive new triples with soc as predicate. With memoization, before the materialization starts a query-driven materialization procedure like QSQR (Abiteboul et al. 1995) or Magic set (Bancilhon et al. 1985), two well-known query-driven algorithms are invoked to compute all answers for the query $T(B, \text{soc}, C)$. Once this procedure is terminated, we can safely assume that the collection of facts that match this atom is immutable; thus the engine can index this collection more efficiently to facilitate the execution of the rule. In some cases, memoization does not lead to any reduction of the materialization runtime, while in other cases, the advantage is significant.

Materialization with Generic Rules

Nonstandard rules are typically easier to execute since they require less joins and predicates have a smaller arity (i.e., they are unary or binary only). On these rules, however, the previous optimizations might not be applicable. Still, the execution can be improved in two ways: Either by applying more general parallel algorithms or by considering multiple facts at the same time. Both types of improvements are described below.

Parallelizing rule execution. Three different types of parallelism can be applied to the rule execution: *Intra-rule parallelism*, *inter-rule parallelism*, and *instance-based parallelism*.

With intra-rule parallelism, the goal is to distribute the execution of a single rule among different processors. For example, let us consider rule (1). In this case, facts that match B_1 could be partitioned into n different partitions depending on the value of the terms that should be joined with B_2 . Similarly, facts that match B_2 could be partitioned in an equivalent number of partitions in so that “ B_1 ” and “ B_2 ” facts with the same join terms will be in the same partition. In this way, each partition can be processed simultaneously by concurrent processors.

With inter-rule parallelism, the idea is to let concurrent processors execute different rules at the same time. For instance, one processor could execute rule (4) while another one execute rule (5). Notice that neither of these two types of parallelism is perfect: With intra-rule parallelism, the computation could be unbalanced if some partitions are much bigger than others. With inter-rule parallelism instead, the maximum number of concurrent processors is bound by the number of rules.

Intra- and inter-based parallelism are well known in literature and are also used in other scenarios. The third type of parallelism is a more recent variant which was first introduced in the RDFox system (Nenov et al. 2015). The idea is to let a number of concurrent processors to continuously pull not-yet-considered facts from a queue and verify whether they instantiate the body of a rule. If this occurs, then the system

searches for other atoms in the database to compute a full rule instantiation. If this process succeeds, then the processor produces a new derivation and puts it back in the queue and database so that it can be further considered, possibly by other processors.

This type of parallelism is significantly different than the other two because here the processors do not receive a predefined amount of work but are free to “steal” computation from each other whenever they become idle. A limitation of this technique is that it requires a number of data structures that allow a fast concurrent access. While hash tables can provide this functionality, they have the disadvantage that they are not cache-friendly, that is, they do not use efficiently the CPU cache.

Set-based rule execution. Another technique for improving the performance consists of generating meta-facts which represents multiple sets of facts. This technique can be intuitively explained with a simple example. Let us consider the rule:

$$P(X, Y) \rightarrow Q(Y, X) \quad (6)$$

and assume that the input database does not contain any q -facts. In this case, it is clear that each fact that matches the body will generate a new q -fact. Thus, the engine can simply create one single fact $q(y^*, x^*)$ where x^* and y^* are special terms which point to set of terms, namely, all first and second terms that appear in p -facts. For instance, if the database equals to $\{p(a, b), p(c, d)\}$, then $y^* \rightarrow \langle a, c \rangle$ and $x^* \rightarrow \langle b, d \rangle$. Notice that the engine does not need to explicitly materialize the lists $\langle a, c \rangle$ and $\langle b, d \rangle$ but can simply store instructions to compute this list on-the-fly in case it is needed.

This technique was first introduced in the VLog system (Urbani et al. 2016), and empirical results show excellent performance against the state of the art, especially because this technique becomes more effective with larger databases as potentially larger sets of facts can be compressed in a single meta-fact.

References

- Abiteboul S, Hull R, Vianu V (1995) Foundations of databases, vol 8. Addison-Wesley, Reading
- Bancilhon F, Maier D, Sagiv Y, Ullman JD (1985) Magic sets and other strange ways to implement logic programs. In: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on principles of database systems. ACM, pp 1–15
- Bizer C, Heath T, Berners-Lee T (2009) Linked data—the story so far. *Int J Semant Web Info Syst* 5(3):1–22
- Brickley D, Guha RV, McBride B (2014) RDF schema 1.1. W3C Recomm 25:2004–2014
- Motik B, Grau BC, Horrocks I, Wu Z, Fokoue A, Lutz C, et al (2009) Owl 2 web ontology language profiles. W3C Recomm 27:61
- Nenov Y, Piro R, Motik B, Horrocks I, Wu Z, Banerjee J (2015) RDFox: a highly-scalable RDF store. In: International semantic web conference. Springer, pp 3–20
- Urbani J, Kotoulas S, Maassen J, Van Harmelen F, Bal H (2012) WebPIE: a web-scale parallel inference engine using mapreduce. *Web Semant Sci Serv Agents World Wide Web* 10:59–75
- Urbani J, Piro R, van Harmelen F, Bal H (2014) Hybrid reasoning on OWL RL. *Semantic Web* 5(6):423–447. <https://doi.org/10.3233/SW-130120>
- Urbani J, Jacobs C, Krötzsch M (2016) Column-oriented datalog materialization for large knowledge graphs. In: Proceedings of AAAI, pp 258–264

- Online machine learning covers methods that update their models after observing a new event and can immediately serve predictions based on the updated model.

Overview

In this chapter, we investigate online learning based recommender algorithms that can efficiently handle nonstationary datasets. We show that online learning for recommendation is rather usual than the exceptional task: For example, if no user history is available, we have to build a user model on the fly, based on the interactions in the live user session.

To the best of our knowledge, this is the first survey with a comprehensive overview of the ideas for recommendation over streaming data and their implementation in various distributed data stream processing systems.

The chapter is based on the notions of online learning, as introduced in the chapter “Overview of Online Machine Learning in Big Data Streams” of this Encyclopedia.

Recommender Systems Over Data Streams

András A. Benczúr¹, Levente Kocsis¹, and Róbert Pálavics²

¹Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary

²Department of Computer Science, Stanford University, Stanford, CA, USA

Synonyms

Incremental learning; Online machine learning on streams; Stream learning

Definitions

- Data stream algorithms process a continuous stream of data with only a limited possibility to store past records.

Introduction

Recommender systems (Ricci et al. 2011) serve to predict user preferences regarding items such as music tracks (Spotify), movies (Netflix), products, books (Amazon), blogs, or microblogs (Twitter), as well as content on friends’ and personal news feeds (Facebook).

Recommenders give a clear, industry-relevant example of the requirements for online machine learning introduced in the Chapter “► Overview of Online Machine Learning in Big Data Streams” of this Handbook. In a typical implementation, users interact with the system by requesting recommendations and then providing feedback by clicking on some of the displayed items. In this way, the users produce a continuous stream of events that can be used for model update and immediate evaluation, for example, by click-through rate. Note that in Žliobaite et al. (2012), it is observed that adaptive learning

models are still rarely deployed in industry. Recommender systems are the main exception: A special class, the session-based recommendation task appears frequently in practice when no past user history is available and user models are built on the fly, using recent interactions in the session.

Recommender systems can be categorized by the type of information they infer about users and items. Collaborative filtering (Linden et al. 2003; Sarwar et al. 2001) builds models of past user-item interactions such as clicks, views, purchases, or ratings, while content-based filtering Lops et al. (2011) recommends items that are similar in content, for example, share phrases in their text description. Context-aware recommenders (Adomavicius and Tuzhilin 2011) use additional information on the user and the interaction, for example, user location and weather conditions. Recent events in a user session (Koenigstein and Koren 2013) serve as a special context.

A milestone in the research of recommendation algorithms, the Netflix Prize competition (Bennett and Lanning 2007), had high impact on research directions. The target of the contest was based on the one- to five-star ratings given by users, with one part of the data used for model training and the other for evaluation. As an impact of the competition, tasks now termed batch rating prediction were dominating research results.

Recommendation models rely on the feedback provided by the user, which can be **explicit**, such as one- to five-star movie ratings on Netflix (Adhikari et al. 2012). However, most recommendation tasks are **implicit**, as the user provides no like or dislike information. Implicit feedback can be available in the form of time elapsed viewing an item or listening to a song, or in many cases, solely as a click or some other form of user interaction. In Pilászy et al. (2015), the authors claim that 99% of recommendation industry tasks are implicit.

As a main difference between recommendation and classification, classifiers usually work independently of the event whose outcome they predict. Recommender systems, on the other

hand, may directly influence observations: They present a ranked top list of items (Deshpande and Karypis 2004), and the user can only provide feedback for the items on the list. Moreover, real systems process data streams where users request one or a few items at a time and get exposed to new information that may change their needs and taste when they return to the service next time. Furthermore, an online trained model may change and return completely different lists for the same user even for interactions very close in time.

By the above considerations, real recommender applications fall in the category of top item recommendation by online learning for implicit user feedback, a task that has received less attention in research so far. In this section, we show the main differences in evaluating such systems compared to both classifiers and batch systems, as well as describe the main data stream recommender algorithms.

Online recommenders seem more restricted than those that can iterate over the data set several times, and one could expect inferior quality from the online methods. By contrast, in Pálovics et al. (2014) and Frigó et al. (2017), surprisingly strong performance of online methods is measured.

As an early time-aware recommender system example, the item-based nearest neighbor (Sarwar et al. 2001) can be extended with time-decay (Ding and Li 2005). Most of the early models, however, are time-consuming to compute and difficult to update from a data stream and hence need periodical batch training. Probably the first result in this area, the idea of processing transactions in chronological order to incrementally train a recommendation model first appeared in Takács et al. (2009, Section 3.5). Streaming gradient descent matrix factorization methods were also proposed in Isaacman et al. (2011) and Ali and Johnson (2011), who use Netflix and MovieLens data and evaluate by root mean square error (RMSE).

The difficulty of evaluating streaming recommenders was first mentioned in Lathia et al. (2009), although the authors evaluated models by offline training and testing split.

Ideas for online evaluation metrics appeared first in Pálovics and Benczúr (2013), Vinagre et al. (2014), and Pálovics et al. (2014). In Vinagre et al. (2014), incremental algorithms are evaluated using recall. In Pálovics et al. (2014), recall is shown to have undesirable properties, and other metrics for evaluating online learning recommenders are proposed.

Finally, we note that batch distributed recommender systems were surveyed in Karydi and Margaritis (2016).

Prequential (Online) Evaluation for Recommenders

To train and evaluate a time-sensitive or online learning recommender, we can use the prequential or online evaluation framework that is described in detail for classifier evaluation in the chapter “► Online Machine Learning Algorithms over Data Streams” of this Handbook. As seen in Fig. 1, online evaluation for a recommender system includes the following steps:

1. We query the recommender for a top- k recommendation for the active user.
2. We evaluate the list in question against the single relevant item that the user interacted with.
3. We allow the recommender to train on the revealed user-item interaction.

Since we can potentially retrain the model after every new event, the recommendation for the same user may be very different even at close points in time, as seen in Fig. 1. The standard

recommender evaluation settings used in research cannot be applied, since there is always only a single relevant item in the ground truth.

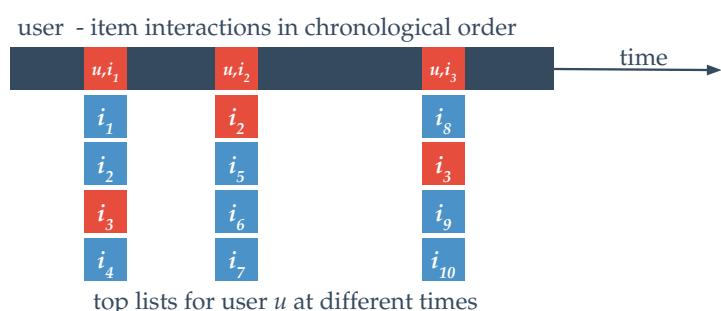
In one of the possible recommender evaluation settings, the **rating prediction** problem, which is popular in research, we consider a user u and an item i . The actual user preference in connection with the item is expressed as a value r_{ui} , for which the system returns a prediction \hat{r}_{ui} . This explicit rating can be a scale such as one to five stars for a Netflix movie, while implicit rating can be the duration of viewing a Web page in seconds. Implicit rating is binary when the only information is whether the user interacted with the item (clicked, viewed, purchased) or not. Depending on whether r_{ui} is binary or scale, the same prequential metrics, such as error rate or mean squared error (MSE), can be applied as for classification or regression. For example, in the Netflix Prize competition, the target was the square root of MSE between the predicted and actual ratings.

Another possible way to evaluate recommenders is **ranking prediction**, where performance metrics depend on the list of displayed items. We note that given rating prediction values \hat{r}_{ui} for all i , in theory, ranking prediction can be solved by sorting the relevance score of all items. For certain models, heuristics to speed up the selection of the highest values of \hat{r}_{ui} by candidate preselection exist (Teflioudi et al. 2015).

To evaluate ranking prediction, we have to take into consideration two issues that do not exist for classifier evaluation. In the case of prequential evaluation, as shown in Fig. 1, the list for user u may change potentially after every

Recommender Systems Over Data Streams, Fig. 1

Prequential evaluation of the online ranking prediction problem



interaction with u . As soon as u provides feedback for certain item i , we can change model parameters and the set of displayed items may change completely. Most of the batch ranking quality measures focus on the set of items consumed by the same user, under the assumption that the user is exposed to the same list of items throughout the evaluation. As this assumption does not hold, we need measures for individual user-item interactions.

Another issue regarding ranking prediction evaluation lies in a potential user-system interaction that affects quality scores. Typically, the set of items is very large, and users are only exposed to a relatively small subset, which is usually provided by the system. The form of user feedback is usually a click on one or more of these items, which can be evaluated by computing the click-through rate. Since users cannot give feedback on items outside the list, the fair comparison of two algorithms that present different sets for the user can only be possible by relying on live user interaction. This fact is known by practitioners, who use **A/B testing** to compare the performance of different systems. In A/B testing, the live set of users is divided into groups that are exposed to the results of the different systems.

Most traditional ranking prediction metrics, to a certain level, rely on the assumption that the same user is exposed to the same list of items, and hence the interactions of the same user can be considered to be the unit for evaluation. For online evaluation, as noted in Pálovics et al. (2014), the unit of evaluation will be a single interaction, which usually contains a single relevant item. Based on this modification, most batch metrics apply in online learning evaluation as well. Note that the metrics below apply not just in A/B testing but also in experiments with frozen data, where user feedback is not necessarily available for the items returned by a given algorithm. For example, if the item consumed by the user in the frozen data is not returned by the algorithm, the observed relevance will be 0, which may not be the case if the same algorithm is applied in an A/B test. Note that attempts to evaluate research results by A/B testing have been made in the information retrieval community (Balog

et al. 2014); however, designing and implementing such experiments is cumbersome.

Next, we list several metrics for the quality of the ordered top- K list of items $L = \{i_1, i_2, \dots, i_K\}$ against the items E consumed by the user. We will also explain how online evaluation metrics differ from their batch counterparts. For the discussion, we mostly follow Pálovics et al. (2014).

Click-through rate is commonly used in the practice of recommender evaluation. It is defined as the ratio of clicks received for L :

$$\text{Clickthrough@K} = \begin{cases} 1 & \text{if } E \cap L \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

For precision and recall, similar to click-through, the actual order within L is unimportant:

$$\begin{aligned} \text{Precision@K} &= \frac{|E \cap L|}{K}, \\ \text{Recall@K} &= \frac{|E \cap L|}{|E|}. \end{aligned} \quad (2)$$

For batch evaluation, E is the entire set of items with positive feedback from a given user who is exposed to the same L for each interaction. The overall batch system performance can be evaluated by averaging precision and recall over the set of users. For online evaluation, typically $|E| = 1$, where Precision@K is 0 or $1/K$ and Recall@K is 0 or 1 depending on whether the actual item in E is listed in L or not. Precision and recall are hence identical to click-through, up to a constant. As a consequence, the properties of online precision and recall are very different from their batch counterparts. The main reason for the difference lies in the averaging procedure of prequential evaluation: We cannot merge the events of the same user; instead, we average over the set of individual interactions.

Measures that consider the position of the relevant item i in L can give more refined performance indication. The first example is reciprocal rank:

$$RR@K = \begin{cases} 0 & \text{if rank}(i) > K; \\ \frac{1}{\text{rank}(i)} & \text{otherwise.} \end{cases} \quad (3)$$

Discounted cumulative gain (DCG) is defined similarly, as

$$\text{DCG@K} = \sum_{k=1}^K \frac{\text{rel}(i_k)}{\log_2(1+k)} \quad (4)$$

where $\text{rel}(i_k)$ indicates the relevance of the i -th item in the list. For the implicit task, relevance is 1 if the user interacted with the item in the evaluation set, 0 otherwise. For batch evaluation, we can consider all interactions of the same user as one unit. If we define iDCG@K , the ideal maximum possible value of DCG@K for the given user, we can obtain nDCG@K , the normalized version of DCG@K, as

$$\text{nDCG@K} = \frac{\text{DCG@K}}{\text{iDCG@K}}. \quad (5)$$

Note that for online learning, there is only one relevant item, hence $\text{iDCG} = 1$. For emphasis, we usually use the name nDCG for batch and DCG for online evaluation.

Session-Based Recommendation

Previous items in user sessions constitute a very important context (Hidasi and Tikk 2016). In e-commerce, the same user may return next time with a completely different intent and may want to see a product category completely different from the previous session. Algorithms that rely on recent interactions of the same user are called **session-based item-to-item** recommenders. The user session is special context, and it is the only information available for an item-to-item recommender. In fact, several practitioners (Koenigstein and Koren 2013; Pilászy et al. 2015) argue that most of the recommendation tasks they face are without sufficient past user history. For example, users are often reluctant to create logins and prefer to browse anonymously. Moreover, they purchase certain types of goods (e.g., expensive electronics) so rarely that their previous purchases will be insufficient to create a meaningful user profile. Whenever a long history of previous activities or purchases by the user is not available,

recommenders may propose items that are similar to the most recent ones viewed in the actual user session.

Session-based recommendation can be served by very simple algorithms, most of which are inherently online. A comparison of the most important such online algorithms in terms of performance is available in Frigó et al. (2017). Data stream processing algorithms can retain items from the most recently started sessions as long as they fit in their memory. Recommendation is based on the recent items viewed by the user in the actual shopping session. For example, we can record how often users visited item i after visiting another item j . Since fast update to transition frequencies is usually possible, the method is online.

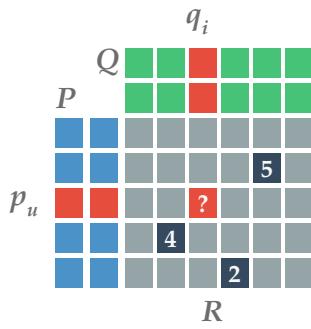
In an even simpler algorithm that is not strictly session-based, we recommend the most popular recent items. This method can be considered batch or online depending on the granularity of the item frequency measurement update. Both algorithms can be personalized if we consider the frequency of past events involving the user. If items are arranged hierarchically (e.g., music tracks by artist and genre), personal popularity and personal session data can involve the frequency of the artists or genres for recommending tracks. More session-based algorithms are described in Koenigstein and Koren (2013).

R

Online Matrix Factorization

Most nontrivial online recommender algorithms are based on matrix factorization (Koren et al. 2009), a popular class of collaborative filtering methods. Given the user-item utility matrix $R = [r_{ui}]$ shown in Fig. 2, we model R by decomposing it into the two dense matrices P and Q . For a given user u , the corresponding row in P is user vector p_u . Similarly, for item i , the corresponding column of Q is item vector q_i . The predicted relevance of item i for user u is then

$$\hat{r}_{ui} = p_u q_i^T. \quad (6)$$



Recommender Systems Over Data Streams, Fig. 2
Utility matrix R and the matrix factorization model built from matrices P and Q

Note that we can extend the above model by scalar terms that describe the biased behavior of the users and the items (Koren et al. 2009).

One possibility to train model parameter matrices P and Q is by gradient descent (Koren et al. 2009; Funk 2006), which can be applied to online learning as well (Pálovics et al. 2014). For a set of interactions E , we optimize Eq. (6) for MSE as target function:

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{r}_{ui} - r_{ui})^2, \quad (7)$$

where r_{ui} is the actual and \hat{r}_{ui} is the predicted rating for user u and item i and N is the current size of the data stream.

In one step of gradient descent, we fit P and Q in Eq. (6) to one of the ratings in E . Unlike in batch training, where we can use the ratings several times in any order, in online learning, we have the most recent single item in E . In other words, in online gradient descent, we fit the model to the events one by one as they arrive in the data stream.

For a given (explicit or implicit) rating r_{ui} , the steps of gradient descent are as follows. First, we compute the gradient of objective function F with respect to the model parameters:

$$\frac{\partial F}{\partial p_u} = -2(r_{ui} - \hat{r}_{ui})q_i, \quad \frac{\partial F}{\partial q_i} = -2(r_{ui} - \hat{r}_{ui})p_u. \quad (8)$$

Next, we update the model parameters in opposite direction of the gradient, proportionally to learning rate η , as

$$p_u \leftarrow \eta(r_{ui} - \hat{r}_{ui})q_i,$$

$$q_i \leftarrow \eta(r_{ui} - \hat{r}_{ui})p_u.$$

Overfitting is usually avoided by adding a regularization term in the objective function (Koren et al. 2009).

In the case of implicit feedback, the known part of the utility matrix only contains elements with positive feedback. To fit a model, one requires negative feedback for training as well. Usually, such elements are selected by sampling from those that the user has not interacted with before Rendle and Freudenthaler (2014). We can also introduce confidence values for ratings and consider lower confidence for the artificial negative events (Hu et al. 2008).

Gradient descent can also be used in a mix of batch and online learning, for example, training batch models from scratch periodically and continuing the training with online learning. We can also treat users and items differently, for example, updating user vectors more dynamically than item vectors, as first suggested by Takács et al. (2009).

Another use of online gradient descent is to combine different recommendation models (Pálovics et al. 2014). We can express the final prediction as the linear combination of the models in the ensemble whose parameters are the linear coefficients and the individual model parameters. In Pálovics et al. (2014), two online gradient descent methods are described with regard to whether the derivative of the individual models is available, where all parameters can be trained through the derivative of the final model or otherwise by learning the coefficients and the individual models separately.

Variants of Matrix Factorization

Several variants of matrix factorization that can be trained by gradient descent both for batch

and online learning tasks have been proposed. Bayesian Personalized Ranking (Rendle et al. 2009) has top list quality as target instead of MSE. In asymmetric matrix factorization (Pátek 2007), we model the user by the sum of the item vectors the user rated in the past.

Recently, various factorization models have been developed that incorporate context information (Hidasi and Tikk 2016). Context data can be modeled by introducing data tensor D instead of the rating matrix R . In a simplest case, the data includes a single piece of additional context information (Rendle and Schmidt-Thieme 2010): for example, music tracks can have artist as context.

Alternating least squares (Koren et al. 2009; Pilászy et al. 2010) (ALS) is another optimization method for matrix factorization models, in which for a fixed Q , we compute the optimal P , then for a fixed P , the optimal Q , repeatedly until certain stopping criteria are met. Hidasi et al. Hidasi and Tikk (2012); Hidasi (2014); Hidasi and Tikk (2016) introduced several variants of ALS-based optimization schemes to incorporate context information. By incremental updating, ALS can also be used for online learning (He et al. 2016).

Conclusions

Recommendation differs from classification in that in recommendation, there are two types of objects, users, and items, and a prediction has to be made for their interaction. A practical recommender system displays a ranked list of a few items for which the user can give feedback. In an online learning system, the list shown to the same user at different times may change completely for two reasons. First, as in the prequential classifier training and evaluation setting described in detail for classifier evaluation in the chapter “► Online Machine Learning Algorithms over Data Streams” of this Handbook, the list of recommendations may change because the model changes. Second, the user feedback we use for evaluation depends on the actual state of the model, since the user may have no means to express interest in an item not displayed. Hence for online learning

evaluation, metrics that involve the notion of a volatile list have to be used.

Online learning, as introduced in the chapter “► Overview of Online Machine Learning in Big Data Streams” of this Handbook, is very powerful for recommender systems due to their advantage of having much more emphasis on recent events. For example, if we update models immediately for newly emerged users and items, trends are immediately detected. The power of online learning for recommendation may also be the result of updating user models with emphasis on recent events, which may be part of the current user session. User session is a highly relevant context for recommendation, and most session-based methods are inherently online.

Cross-References

- Online Machine Learning Algorithms over Data Streams
- Overview of Online Machine Learning in Big Data Streams
- Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning

Acknowledgements Support from the EU H2020 grant Streamline No 688191 and the “Big Data—Momentum” grant of the Hungarian Academy of Sciences.

References

- Adhikari VK, Guo Y, Hao F, Varvello M, Hilt V, Steiner M, Zhang ZL (2012) Unreeling netflix: understanding and improving multi-cdn movie delivery. In: INFOCOM, 2012 Proceedings IEEE. IEEE, pp 1620–1628
- Adomavicius G, Tuzhilin A (2011) Context-aware recommender systems. In: Ricci F, Rokach L, Shapira B, Kantor PB (eds) Recommender systems handbook. Springer, Boston, pp 217–253
- Ali M, Johnson CC, Tang AK (2011) Parallel collaborative filtering for streaming data. University of Texas Austin, Technical Report
- Balog K, Kelly L, Schuth A (2014) Head first: living labs for ad-hoc search evaluation. In: Proceedings of the 23rd ACM international conference on conference on information and knowledge management. ACM, pp 1815–1818
- Bennett J, Lanning S (2007) The netflix prize. In: KDD Cup and workshop in conjunction with KDD 2007

- Deshpande M, Karypis G (2004) Item-based top-n recommendation algorithms. *ACM Trans Inf Syst (TOIS)* 22(1):143–177
- Ding Y, Li X (2005) Time weight collaborative filtering. In: Proceedings of the 14th ACM international conference on Information and knowledge management. ACM, pp 485–492
- Frigó E, Pálovics R, Kelen D, Benczúr AA, Kocsis L (2017) Online ranking prediction in non-stationary environments. In: Proceedings of the 1st workshop on temporal reasoning in recommender systems, co-located with 11th international conference on recommender systems
- Funk S (2006) Netflix update: try this at home. <http://sifter.org/simon/journal/20061211.html>
- He X, Zhang H, Kan MY, Chua TS (2016) Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval. ACM, pp 549–558
- Hidasi B (2014) Factorization models for context-aware recommendations. *Infocommun J VI*(4):27–34
- Hidasi B, Tikk D (2012) Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In: Machine learning and knowledge discovery in databases. Springer, pp 67–82
- Hidasi B, Tikk D (2016) General factorization framework for context-aware recommendations. *Data Min Knowl Discov* 30(2):342–371
- Hu Y, Koren Y, Volinsky C (2008) Collaborative filtering for implicit feedback datasets. In: Eighth IEEE international conference on data mining, 2008. ICDM'08. IEEE, pp 263–272
- Isaacman S, Ioannidis S, Chaintreau A, Martonosi M (2011) Distributed rating prediction in user generated content streams. In: Proceedings of the fifth ACM conference on recommender systems. ACM, pp 69–76
- Karydi E, Margaritis K (2016) Parallel and distributed collaborative filtering: a survey. *ACM Comput Surv (CSUR)* 49(2):37
- Koenigstein N, Koren Y (2013) Towards scalable and accurate item-oriented recommendations. In: Proceedings of the 7th ACM conference on recommender systems. ACM, pp 419–422
- Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37
- Lathia N, Hailes S, Capra L (2009) Temporal collaborative filtering with adaptive neighbourhoods. In: Proceedings of the 32nd international ACM SIGIR conference on research and development in information retrieval. ACM, pp 796–797
- Linden G, Smith B, York J (2003) Amazon.com recommendations: item-to-item collaborative filtering. *Internet Comput IEEE* 7(1):76–80
- Lops P, De Gemmis M, Semeraro G (2011) Content-based recommender systems: state of the art and trends. In: Ricci F, Rokach L, Shapira B, Kantor, PB (eds) Recommender systems handbook. Springer, Boston, pp 73–105
- Pálovics R, Benczúr AA (2013) Temporal influence over the Last.fm social network. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining. ACM, pp 486–493
- Pálovics R, Benczúr AA, Kocsis L, Kiss T, Frigó E (2014) Exploiting temporal influence in online recommendation. In: Proceedings of the 8th ACM conference on recommender systems. ACM, pp 273–280
- Paterek A (2007) Improving regularized singular value decomposition for collaborative filtering. In: Proceedings of KDD Cup workshop at SIGKDD'07, 13th ACM international conference on knowledge discovery and data mining, pp 39–42
- Pilászy I, Serény A, Dózsa G, Hidasi B, Sári A, Gub J (2015) Neighbor methods vs matrix factorization – case studies of real-life recommendations. In: LSRS2015 at RECSYS
- Pilászy I, Zibriczky D, Tikk D (2010) Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In: Proceedings of the fourth ACM conference on recommender systems. ACM, pp 71–78
- Rendle S, Freudenthaler C (2014) Improving pairwise learning for item recommendation from implicit feedback. In: Proceedings of the 7th ACM international conference on web search and data mining. ACM, pp 273–282
- Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2009) Bpr: Bayesian personalized ranking from implicit feedback. In: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence. AUAI Press, pp 452–461
- Rendle S, Schmidt-Thieme L (2010) Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the third ACM international conference on web search and data mining. ACM, pp 81–90
- Ricci F, Rokach L, Shapira B (2011) Introduction to recommender systems handbook. In: Ricci F, Rokach L, Shapira B, Kantor PB (eds) Recommender systems handbook. Springer, Boston
- Sarwar B, Karypis G, Konstan J, Reidl J (2001) Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th international conference on World Wide Web (WWW'01). ACM Press, New York, pp 285–295. <https://doi.org/10.1145/371920.372071>. <http://portal.acm.org/citation.cfm?id=372071>
- Takács G, Pilászy I, Németh B, Tikk D (2009) Scalable collaborative filtering approaches for large recommender systems. *J Mach Learn Res* 10:623–656
- Teflioudi C, Gemulla R, Mykytiuk O (2015) Lemp: fast retrieval of large entries in a matrix product. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 107–122
- Vinagre J, Jorge AM, Gama J (2014) Evaluation of recommender systems in streaming environments. In: Workshop on recommender systems evaluation: dimensions and design (REDD 2014), held in conjunction with RecSys 2014, Silicon Valley, Oct 10, 2014

Žliobaite I, Bifet A, Gaber M, Gabrys B, Gama J, Minku L, Musial K (2012) Next challenges for adaptive learning systems. ACM SIGKDD Explor News 14(1): 48–55

Record Linkage

Anja Gruehnheid
Google Inc., Madison, WI, USA

Synonyms

Duplicate detection; Entity resolution

Definitions

Record linkage refers to the task of extracting record information from various input data sources and combining them in such a way that each output record corresponds to a distinct real-world entity.

Overview

Record linkage is part of the broader area of data integration and more specifically data cleaning. It is most commonly used as a means to identify duplicates in a dataset or multiple datasets. The biggest challenge when executing record linkage algorithms is the trade-off between quality and performance. That is, record linkage is often run on high-volume datasets for which even sophisticated algorithms will not be able to provide high-quality results in a suitable timeframe. Thus, techniques such as blocking or incremental computation are applied to improve performance at the cost of decreased result quality. Additional challenges in record linkage include various types of input sources that are not necessarily structured. For example, human-generated data for record linkage has been extensively studied in recent years under the assumption that human-generated data on the similarity of records has better quality than machine-generated data.

Key Research Findings

Record linkage is a decades-old problem that has been studied extensively. It can be split into several key research areas. First, *similarity computation* describes the challenge of identifying whether two records are similar. Second, there exists a variety of *algorithms* that take these similarities as input and output (sets of) records where each record represents a real-world entity. Finally, *performance* challenges have arisen in the context of record linkage due to an increase in data volume and velocity and the inherent quadratic scaling of exhaustive record linkage algorithms.

Similarity Computation

To identify whether two records refer to the same entity, it is crucial to measure their similarity. For example, Table 1 shows different business addresses that can be found in datasets obtained from (semi-)manually curated business listings. Here, different records r_i have different formats, some have erroneous values, and others may have missing data. This exemplifies that comparable similarity computation is not trivial. For example, is r_1 as similar to r_2 (different formatting) as it is to r_3 (wrong phone number)? The most common similarity computation techniques are based on the similarity of the characters in the record strings, Elmagarmid et al. (2007), when comparing two input records. Examples for pair-wise character-based similarity computation techniques are edit distances such as the Levenshtein distance, Levenshtein (1966), or metrics such as the Jaro similarity metric, Jaro (1978). Alternatively, some record linkage systems also use token-based similarity, first proposed by Monge et al. (1996), or phonetic similarity, Russell (1922). The chosen similarity metric often depends on the use case that the record linkage mechanism is applied in.

Given these similarity metrics, it is obvious that resolving the similarity between records r_i and r_j is not always certain. That is, independent of the actual metric, the pair-wise similarity of records r_i and r_j is typically represented as a probability $p \in [0, 1]$ where $p(r_i, r_j) = 1$ signi-

Record Linkage, Table 1 Example business addresses

ID	Name	Address	City	Phone
r_1	Peet's coffee	2124 Vine St Ste1	Berkeley	(510) 841-0564
r_2	Peet's coffee	2124 VINE ST	BERKELEY	5108410564
r_3	Peet's coffee	2124 Vine St	Berkeley	5102257700
r_4	Peet's	2501 Telegraph Ave	Berkeley	(510) 225-7700
r_5	Peet's	2501 Telegraph Ave		(510) 225-7700
r_6	Peet's coffee		Berkeley	

fies that both records are identical. Uncertainty in the similarity computation can thus be expressed explicitly, i.e., $p(r_i, r_j) \approx 0.5$.

Algorithms

Algorithms for record linkage take as input the pair-wise similarity values, evaluate them, and output a partitioning of records $R = \bigcup R_k$. In this partitioning, each record $r_i^k \in R_k$ points to the same real-world entity e_k . This problem has been approached from various angles. Amongst the most significant are techniques such as *probabilistic modeling*, *graph clustering*, and *(semi-)supervised learning* described in detail next. Note that this is not a complete list of record linkage algorithms many of which are explained and evaluated by Hassanzadeh et al. (2009) or discussed in Dong and Srivastava (2015).

Probabilistic Modeling. First formalized by Fellegi and Sunter (1969), the core idea of probabilistic modeling for record linkage is to assign a record pair $\gamma = (r_i, r_j)$ to either a set of matches M or non-matches U based on the computational similarity of γ . Specifically, assigning the pair to M signifies that $r_i = r_j$, while if the pair is in U , then $r_i \neq r_j$ holds. The record pair is assigned to M if the probability of assigning γ to M based on the agreement $m(\gamma)$ is higher than the probability of assigning γ to U based on disagreement $u(\gamma)$, i.e., $P(M|m(\gamma)) > P(U|u(\gamma))$. Using the Bayes rule, this can be rewritten as

$$\gamma \in \begin{cases} M, & \text{if } \frac{P(m(\gamma)|M)}{P(u(\gamma)|U)} > \frac{P(U)}{P(M)}, \\ U & \text{otherwise.} \end{cases} \quad (1)$$

Probabilistic modeling assumes, as the name indicates, that the decision whether γ is a match is non-trivial and has to be expressed as a likelihood. Thus, by design, there will be cases where there exists evidence for γ being in M and U at the same time. To resolve this problem, probabilistic modeling assumes that assigning γ to M even though there exists some evidence that it should be in U incurs an error reflected in a penalty. Finding the assignment of all possible record pairs thus becomes an *error minimization* problem. Furthermore, Fellegi and Sunter (1969) observed that the similarity functions that determine the agreements and disagreements of γ may contain errors. Thus, they loosened their categorization of γ to allow it to be classified as a *possible pair* if there is evidence for both a match and a non-match. Pairs in that group are confirmed by human workers after the automated record linkage has concluded.

Correlation Clustering. Correlation clustering is a popular graph clustering technique for record linkage, Bansal et al. (2004). At its core, it builds upon the idea of probabilistic modeling to determine whether a record pair is a match or non-match, but instead of assigning the pair to distinct classes, correlation clustering assigns them to distinct record clusters where each cluster represents a real-world entity. The assignment is done analogous to the idea of error minimization in probabilistic modeling, i.e., if the penalty of adding a record r_i to a cluster c_k is smaller than the penalty of adding it to any other cluster or keeping it as a singleton cluster, then the record is added to c_k . Note that correlation clustering can also be formulated as an *agreement maximization*

problem which may be easier to implement for some use cases.

Formally, correlation clustering works as follows. Assume that F is the penalty for the a clustering C and $p(r_i, r_j)$ is the probability of r_i and r_j belonging to the same cluster $c_k \in C$. An optimal clustering C^* is the clustering that has the smallest value of F , i.e., F^* , given any possible clustering. Recall that its value is computed as the global penalty. In other words, if records r_i and r_j are in the same cluster c_k but $p(r_i, r_j) < 1$, then this clustering incurs a penalty of $1 - p(r_i, r_j)$. Similarly, if r_i and r_j are not in c_k , the clustering incurs a penalty of $p(r_i, r_j)$. F is thus computed as:

$$F(C) = \sum_{c_k \in C \wedge r_i, r_j \in c_k} 1 - p(r_i, r_j) + \sum_{c_k \in C \wedge r_i \in c_k \wedge r_j \notin c_k} p(r_i, r_j) \quad (2)$$

Note that there may exist multiple clusterings that all have the same value F^* and are thus considered optimal.

Calculating C^* exhaustively is computationally infeasible. Thus, Bansal et al. (2004) propose an approximation algorithm called *cautious* correlation clustering with an approximation parameter δ . It proceeds iteratively as follows on a set of records R :

1. Pick a node $r_i \in R$ at random.
2. Add the neighborhood of r_i , i.e., all records connected to r_i , to cluster c_k .
3. Remove all records in c_k that are not 3δ -good.
4. Add all records in the neighborhood of c_k that are 7δ -good.
5. Remove all records in c_k from R . Jump to 1.

The parameter δ is specific to cautious correlation clustering and is used to give a guarantee of how much the approximated solution diverges from the optimal solution. Specifically, the authors show that with the above algorithm, a $9(\frac{1}{\delta^2} + 1)$ approximation can be found in $O(R^2)$.

(Semi-)Supervised Learning. Learning techniques have shown potential for improvement of record linkage solutions especially for record linkage scenarios where the similarity of records

cannot be captured with simpler similarity comparison techniques such as character-based similarities. Amongst others, there has been work on applying SVMs, Bilenko et al. (2003), and learning how to cluster records that refer to the same real-world entity in a supervised manner, Cohen and Richman (2002). More recently, research in the area of record linkage has seen an increase in work combining crowdsourcing, i.e., human responses, and automated entity resolution techniques. For example, Wang et al. (2012) and as follow-up Wang et al. (2014) discuss strategies to minimize the number of crowd requests while at the same time increasing the quality of the record linkage solution. Specifically, the idea in this line of work is to leverage positive responses of the crowd workers to reduce the search space based on transitivity. That is, if records r_i and r_j represent the same real-world entity, and r_j and r_l refer to different entities, then the crowd does not need to confirm the relationship of r_i and r_l . Other work in this area has also utilized similar techniques as applied for probabilistic modeling to maximize the quality of a crowdsourced record linkage solution, Verroios and Garcia-Molina (2015).

Performance Improvements

With an increase in stored data and processing capabilities, new challenges have arisen for executing traditional record linkage efficiently. For example, large datasets in the 1960s for which some of the original record linkage algorithms were developed, are multiple orders of magnitude smaller than current linkage datasets. Thus, several performance improvement techniques such as *blocking* or *incremental data processing* have been applied in the context of record linkage to make this process computationally feasible. A drawback of these algorithms and mechanisms is that they often trade off quality for performance.

Blocking. The purpose of blocking is to parallelize computationally expensive steps in the record linkage process. Basically, blocking partitions the record set R according to a pre-defined input function f that maps a record $r_i \in R$ to a block B_k , Jaro (1989). The goal of f is to

map all records that point to the same real-world entity to the same B_k . Techniques such as similarity comparison which would have to be run in $O(R^2)$ before can then be run in $O(|B_k|^2)$. The applied mapping function is either based on the record characteristics such as the words, tokens, or k-grams it contains, domain knowledge of the record linkage context, user-specified constraints, Arasu et al. (2009), or techniques such as sorted neighborhood, Hernández and Stolfo (1998). The drawback of defining such functions is that there are no guarantees whether any blocking function will correctly map all records of the same real-world entity to the same block. Approaches to address this problem are, for example, to deploy multiple blocking functions or compute overlapping canopies instead of distinct blocks, McCalum et al. (2000).

Incremental Record Linkage. Datasets that are deduplicated are often static and only occasionally updated. The idea behind incremental record linkage is to leverage that observation and to only (re)compute those parts of the record linkage solution that have been directly or indirectly modified by updates in the input dataset. First discussed by Benjelloun et al. (2009) as agglomerative clustering, the basic idea of incremental record linkage was extended by Gruenheid et al. (2014) to allow any kind of data modification of the input dataset and to provide theoretical guarantees for graph record linkage techniques.

In practice, incremental record linkage does not modify the idea of the original batch linkage algorithm, for example, correlation clustering, but modifies these algorithms to fit an iterative linkage use case. Depending on the applied batch linkage algorithm, this may lead to a decrease in the quality of the linked results if the record linkage computation is global and cannot be localized, i.e., when the whole dataset is required as input to form an optimal solution.

Applications of Record Linkage

Entity resolution is a task important for many different data integration and more recently ma-

chine learning systems. Thus, there exist dedicated systems such as IBM InfoSphere that have been commercializing record linkage for many decades as part of so-called ETL (extract, transform, load) processes. The most common use case for commercial system is that legacy data in companies is stored in various places such as databases, file systems, etc. and needs to be combined efficiently and without losing any of the input data. Deduplication issues arise also if companies merge and the data of the acquired company has to be integrated into the existing internal database. Next to commercial systems, there also exist open source alternatives that can be used for entity resolution such as Konda et al. (2016).

Furthermore, record linkage is part of pipelines used for data cleaning. The goal is to find “dirty” data which is the same record inserted into the system multiple times with slight variations; see Table 1 for an example. It is crucial to have good-quality linkage results for such pipelines as the consolidated entities are often shown in user interfaces. For example, information about restaurants, attractions, etc. can be obtained from a number of input data sources such as the place’s website, community boards discussing the place, or third parties storing information on businesses. Thus, it needs to be consolidated before appearing on platforms such as Facebook or Google Maps.

Future Directions for Research

Although record linkage has been extensively studied, developments in data processing and learning continuously change its scope. For example, with the construction of knowledge bases, record linkage became a focus of those that wanted to interpret user feedback and content. They then developed new techniques for the new context in which record linkage was applied in. Next to specific applications of record linkage, there also exists a general trend to make record linkage part of existing data integration pipelines and to provide tools that enable users to execute record linkage with low overhead and in near

real-time. As a result, one of the challenges and still an open problem of such an integration is interactive record linkage given that most traditional linkage algorithms have been designed for offline processing.

Cross-References

► Large-Scale Entity Resolution

References

- Arasu A, Ré C, Suciu D (2009) Large-scale deduplication with constraints using dedupalog. In: Proceedings of the 25th international conference on data engineering, ICDE, 29 Mar–2 Apr 2009, Shanghai, pp 952–963. <https://doi.org/10.1109/ICDE.2009.43>
- Bansal N, Blum A, Chawla S (2004) Correlation clustering. *Mach Learn* 56(1–3):89–113
- Benjelloun O, Garcia-Molina H, Menestrina D, Su Q, Whang SE, Widom J (2009) Swoosh: a generic approach to entity resolution. *VLDB J Int J Very Large Data Bases* 18(1):255–276
- Bilenko M, Mooney R, Cohen W, Ravikumar P, Fienberg S (2003) Adaptive name matching in information integration. *IEEE Intell Syst* 18(5):16–23
- Cohen WW, Richman J (2002) Learning to match and cluster large high-dimensional data sets for data integration. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 475–480
- Dong XL, Srivastava D (2015) Big data integration. *Synth Lect Data Manag* 7(1):1–198
- Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate record detection: a survey. *IEEE Trans Knowl Data Eng* 19(1):1–16
- Fellegi IP, Sunter AB (1969) A theory for record linkage. *J Am Stat Assoc* 64(328):1183–1210
- Gruenheid A, Dong XL, Srivastava D (2014) Incremental record linkage. *Proc VLDB Endow* 7(9):697–708
- Hassanzadeh O, Chiang F, Miller RJ, Lee HC (2009) Framework for evaluating clustering algorithms in duplicate detection. *VLDB* 2(1):1282–1293
- Hernández MA, Stolfo SJ (1998) Real-world data is dirty: data cleansing and the merge/purge problem. *Data Min Knowl Disc* 2(1):9–37
- Jaro MA (1978) Unimatch: a record linkage system: users manual. Bureau of the Census, Washington DC
- Jaro MA (1989) Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J Am Stat Assoc* 84(406):414–420
- Konda P, Das S, C PSG, Doan A, Ardalan A, Ballard JR, Li H, Panahi F, Zhang H, Naughton JF, Prasad S, Krishnan G, Deep R, Raghavendra V (2016) Magellan: toward building entity matching management systems. *VLDB* 9(12):1197–1208. <http://www.vldb.org/pvldb/vol9/p1197-pkonda.pdf>
- Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics Doklady, vol 10, pp 707–710
- McCallum A, Nigam K, Ungar LH (2000) Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 169–178
- Monge AE, Elkan C et al (1996) The field matching problem: algorithms and applications. In: KDD, pp 267–270
- Russell R (1922) Index. US Patent 1,435,663. <https://www.google.com/patents/US1435663>
- Verroios V, Garcia-Molina H (2015) Entity resolution with crowd errors. In: 31st IEEE international conference on data engineering, ICDE 2015, Seoul, 13–17 Apr 2015, pp 219–230. <https://doi.org/10.1109/ICDE.2015.7113286>
- Wang J, Kraska T, Franklin MJ, Feng J (2012) Crowd: crowdsourcing entity resolution. *Proc VLDB Endow* 5(11):1483–1494
- Wang J, Li G, Kraska T, Franklin MJ, Feng J (2014) Leveraging transitive relations for crowdsourced joins. CoRR abs/1408.6916. <http://arxiv.org/abs/1408.6916>

R

Regular Path Queries

► Graph Path Navigation

Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning

András A. Benczúr¹, Levente Kocsis¹, and Róbert Pálavics²

¹Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary

²Department of Computer Science, Stanford University, Stanford, CA, USA

Synonyms

Incremental learning; Online machine learning on streams; Stream learning

Definitions

- Data stream algorithms process a continuous stream of data with only a limited possibility to store past records.
- Online machine learning covers methods that update their models after observing a new event and can immediately serve predictions based on the updated model.

Overview

In this chapter, we give a brief overview the following special topics in online machine learning: Reinforcement learning; unsupervised data mining methods, including clustering, frequent itemset mining, dimensionality reduction, and topic modeling; finally, we list the most important concept drift adapting learning methods.

This Chapter is an extension of the other chapters in this Handbook, “Overview of Online Machine Learning in Big Data Streams”, “Online Machine Learning Algorithms over Data Streams”, and “Recommender systems over Data Streams”

Reinforcement Learning

Reinforcement learning is an area of machine learning concerned with agents taking actions in an environment with the aim of maximizing some cumulative reward. It is different from supervised learning in that the environment does not provide a target behavior, only rewards depending on the actions taken.

The environment is typically assumed to be a Markov decision process (MDP). Formally, we assume a set of states, S ; a set of actions, A ; and a transition probability function $P(s, a, s')$ denoting the probability of reaching state s' after taking action a in state s and a reward function $R(s, a)$ denoting the immediate reward after taking action a in state s .

While there is a wide range of reinforcement learning algorithms (see, e.g., Sutton and Barto 1998), we focus here on algorithms that fit the streaming model and (possibly) deal with non-stationary environments. The streaming model of

reinforcement learning is constrained not only by a continuous flow of input data but also by a continuous requisite to take actions.

Algorithms for Stationary Environments

Most reinforcement learning algorithms estimate the value of feasible actions and build a policy based on that value (e.g., by choosing the actions with the highest estimates with some additional exploration). An alternative to value prediction methods are policy gradient methods that update a parameterized policy depending on the performance.

Value Prediction

The value of a state is the expected cumulative reward starting from a given state and following a particular policy. In a similar way, the action value is the expected reward starting from a given state with a particular action.

Value prediction methods estimate the value of the state or the value of the actions in particular states. In the former case, to build a policy from the estimated values, an additional transition model is needed as well. Such a model is provided for some domains (e.g., by the rules of a game), but in many cases, the transition model needs to be learned as well. Action values can be used directly for constructing a policy without the need for a model.

Temporal difference (TD) learning learns the state value estimate $V(s)$ by the following update rule after each state transition (S_t, S_{t+1}) :

$$V(S_t) \leftarrow (1 - \alpha)V(S_t) + \alpha(R_t + \gamma V(S_{t+1})),$$

where α is a step-size and γ is the discount factor. TD learning was used in one of the first breakthroughs for reinforcement learning, that is, Tesauro’s backgammon program (Tesauro 1995).

The best-known action-value prediction algorithm is Q-learning (Watkins and Dayan 1992). For each occurrence of a transition (S_t, A_t, S_{t+1}) , the algorithm updates the action-value $Q(S_t, A_t)$ by

$$\begin{aligned} Q(S_t, A_t) &\leftarrow (1 - \alpha)Q(S_t, A_t) \\ &+ \alpha(R_t + \gamma \max_a Q(S_{t+1}, a)). \end{aligned}$$

Q-learning using deep neural network to approximate the action-values has been successfully applied to playing some Atari games at human expert level (Mnih et al. 2015).

Another algorithm that learns action-values is Sarsa (Sutton 1996). For each sequence $S_t, A_t, S_{t+1}, \text{and } A_{t+1}$, the algorithm updates its estimates by

$$\begin{aligned} Q(S_t, A_t) &\leftarrow (1 - \alpha)Q(S_t, A_t) \\ &+ \alpha(R_t + \gamma Q(S_{t+1}, A_{t+1})). \end{aligned}$$

Sarsa was successfully used by Ipek et al. (2008) for optimizing a DRAM memory controller.

The value prediction algorithms above were described with update rules for a tabular representation. In most cases, function approximation is used, and the update rules rely on a gradient step. Online enhancements of gradient descent as well as eligibility traces (Sutton and Barto 1998) can be applied to all variants.

Policy Gradient

While using value functions is more widespread, it is also possible to use a parameterized policy without relying on such functions. Parameterized policies are typically optimized by gradient ascent with respect to the performance of the policy.

A policy gradient algorithm, the REINFORCE algorithm (Williams 1992), was used to optimize policy in a Go playing program that outperforms the best human players (Silver et al. 2016).

Algorithms for Nonstationary Environments

Most reinforcement learning algorithms, including those discussed in the previous section, assume that the environment does not change over time. While incremental algorithms such as Q-learning can adapt well to nonstationary environments, it may be necessary to devise more explicit exploration strategies that can cope with changes, for example, in reward distribution.

A special case of reinforcement learning is the multiarmed bandit problem. In this case, the

agent repeatedly selects an action from K possible choices, obtaining a reward after each choice. This problem retains the notion of reward; however, there are no states and consequently no state transitions. In the non-stochastic variant (Auer et al. 2002), the distribution of the rewards may change over time arbitrarily. Standard algorithms for this problem are Exp3 and its variants (Auer et al. 2002), which rely on an exponential selection algorithm, including some exploration terms as well. Contextual bandits extend the bandit setting with the notion of state (or context); however, state transitions are still missing. This framework was used, for instance, in (Li et al. 2010) to select personalized new stories. We note that the distinguishing feature of recommendation in a bandit setting is that the user can provide feedback only on the recommended items.

Unsupervised Data Mining

The most prominent class of unsupervised learning methods is **clustering** where instances have to be distributed into a finite set of clusters such that instances within the cluster are more similar to each other than to others in different clusters (Pang-Ning 2006). Batch clustering algorithms have been both studied and employed as data analysis tools for decades (Jain et al. 1999; Wunsch 2008). One frequently applied clustering method is **k-means** (Hartigan and Hartigan 1975) where cluster center selection and assignment to nearest centers are iteratively performed until convergence. Another is **DBSCAN** (Ester et al. 1996), a density-based method that groups points that are closely packed together.

Online clustering algorithms are surveyed among other places in Mahdiraji (2009), Kavitha and Punithavalli (2010), Aggarwal (2013), and Silva et al. (2013). The majority of the most relevant methods are data stream versions of k-means or its variants such as k-medians (Zhang et al. 1996; Bradley et al. 1998; Farnstrom et al. Farnstrom; O'callaghan et al. 2002; Guha et al. 2003; Aggarwal et al. 2003; Zhou et al. 2008; Gama et al. 2011; Kranen et al. 2011; Ackermann et al. 2012). Another set of results describes the

data stream implementation of DBSCAN (Cao et al. 2006; Chen and Tu 2007; Kranen et al. 2011). Finally, an online hierarchical clustering algorithm that maintains similarity measures and hierarchically merges closest clusters is described in Rodrigues et al. (2006).

Finding frequent itemsets Agrawal et al. (1993) is another central unsupervised data mining task, both static and streaming. In brief, for a table of transactions and items, the task is to find all subsets of items that occur together in transactions with at least a prescribed frequency. Several variants of the task are described in Aggarwal and Han (2014). Online frequent itemset mining algorithms are surveyed in Cheng et al. (2008b). Algorithms based on counts of all past data in the stream (Chang and Lee 2003; Giannella et al. 2003; Li et al. 2004; Yu et al. 2004; Lee and Lee 2005) are also called landmark window-based approaches. In some of these algorithms, time adaptivity is achieved by placing more importance on recent items (Chang and Lee 2003; Giannella et al. 2003; Lee and Lee 2005). Sliding window-based approaches (Chang and Lee 2003; Chi et al. 2006; Chang and Lee 2006; Song et al. 2007; Cheng et al. 2008a; Li et al. 2009; Yen et al. 2011; Calders et al. 2014) are particularly suitable for processing data with concept drift. For a comparative overview, see, for example, how MOA’s algorithm was selected (Quadrana et al. 2015). Note that a special subtask, finding frequent items in data streams, is already challenging and requires approximate data structures (Charikar et al. 2004).

Principal component analysis (PCA) is a powerful tool for dimensionality reduction (Jolliffe 1986) based on matrix factorization. Online variants are based on ideas to incrementally update the matrix decomposition (Bunch and Nielsen 1978; Hall et al. 2000; Brand 2002). The first PCA algorithms suitable for online learning are based on neural networks (Oja 1982; Sanger 1989; Oja 1992). Similar to linear classification and regression models, PCA can also apply the kernel trick to involve nonlinear modeling (Schölkopf et al. 1998). Iterative **kernel PCA** is described in Kim et al. (2005) and Günter

et al. (2007) and online kernel PCA in Honeine (2012). We note that for nearest neighbor search in the low-dimensional space provided by PCA, the heuristics for selecting large inner products is applicable (Teflioudi et al. 2015).

Probabilistic topic modeling fits complex hierarchical Bayesian models to large document collections. A topic model reveals latent semantic structure that can be used for many applications. While PCA-like models can also be used for latent semantic analysis (Deerwester et al. 1990), recently the so-called **Latent Dirichlet Allocation (LDA)** (Blei et al. 2003) has gained popularity. Most topic model parameters can only be inferred based on Markov Chain Monte Carlo sampling, a method difficult to implement for online learning. LDA inference is possible based on either online Gibbs sampling (Song et al. 2005; Canini et al. 2009) or online stochastic optimization with a natural gradient step (Hoffman et al. 2010). Several online LDA variants are described in Smola and Narayananurthy (2010), Ho et al. (2013), Li et al. (2014), Yuan et al. (2015), Yu et al. (2015), Jagerman et al. (2017).

Concept Drift and Adaptive Learning

In dynamically changing and nonstationary environments, we often observe concept drift as the result of data distribution change over time. The phenomenon and mitigation of concept (or dataset) drift for online learning are surveyed in several articles (Widmer and Kubat 1996; Tsymbal 2004; Quionero-Candela et al. 2009; Žliobaite et al. 2012; Gama et al. 2014). The area of transfer learning where the (batch) training and the test sets are different (Pan and Yang 2010) is closely related to concept drift (Storkey 2009) but more difficult in the sense that adaptation by learning part of the new data is not possible.

Adaptive learning refers to the technique of updating predictive models online to react to concept drifts. One of the earliest active learning systems is STAGGER (Schlimmer and Granger 1986). In Žliobaitė (2009), the main steps of online adaptive learning are summarized as (1) making assumptions about future distribution, (2)

identifying change patterns, (3) designing mechanisms to make the learner adaptive, and (4) parameterizing the model at every time step.

A comprehensive categorization of concept drift adaptation techniques is found in Gama et al. (2014). Online learning algorithms can naturally adapt to evolving distributions. However, adaptation happens only as the old concepts are diluted due to the new incoming data, which is more suitable for gradual changes (Littlestone 1988; Domingos and Hulten 2000). For sudden changes, algorithms that maintain a sliding window of the last seen instances perform better (Widmer and Kubat 1996; Gama et al. 2004; Kuncheva and Žliobaitė 2009). Another option is to include explicit forgetting mechanisms (Koychev 2000; Klinkenberg 2004; Elwell and Polikar 2009). The most important distinction is whether changes are explicitly or implicitly detected: **Trigger-based** methods aim at detecting when concept drift occurs to build a new model from scratch (Gama et al. 2004). **Evolving learners**, by contrast, do not aim to detect changes but rather maintain the most accurate models at each time step. Evolving learners are method-specific, most of them are based on ensemble methods (Wang et al. 2003; Kolter and Maloof 2003).

A few papers (Minku et al. 2010; Moreno-Torres et al. 2012) give overviews of different types of environmental changes and concept drifts based on speed, recurrence, and severity. Drift can happen gradually or suddenly, in isolation, in tendencies or seasonally, and predictably or unpredictably, and its effect on classifier performance may or may not be severe. In Schlimmer and Granger (1986), Gama et al. (2004), several artificial data sets with different drift concepts, sudden or abrupt, and gradual changes are described.

A large variety of single classifier and ensemble models capable of handling concept drift are described in Tsymbal (2004). Perhaps the majority of the results consider tree-based methods Alberg et al. (2012). For example, concept drift adaptive online decision trees based on a statistical change detector that works on sliding windows are described in Bifet and Gavald (2009),

Bifet (2010). More examples include Bayesian models (Gama et al. 2003; Bach and Maloof 2010), neural networks (Gama and Rodrigues 2007; Leite et al. 2013), and SVM (Syed et al. 1999; Klinkenberg and Joachims 2000). Concept drift adaptation methods exist for clustering (Rodrigues et al. 2006; Silva et al. 2013). Sliding window-based data stream frequent itemset mining is also adaptive (Quadrana et al. 2015). Some of the results do not follow the data stream computational model but rather use computational resources with little restriction. One class of such methods are incremental algorithms with partial memory (Maloof and Michalski 2004). We also note that there is a MOA-based software system for concept drift detection (Bifet et al. 2013).

Cross-References

- ▶ [Overview of Online Machine Learning in Big Data Streams](#)
- ▶ [Online Machine Learning Algorithms over Data Streams](#)
- ▶ [Recommender systems over Data Streams](#)

Acknowledgements Support from the EU H2020 grant Streamline No 688191 and the “Big Data—Momentum” grant of the Hungarian Academy of Sciences.

References

- Ackermann MR, Märtens M, Raupach C, Swierkot K, Lammersen C, Sohler C (2012) Streamkm++: a clustering algorithm for data streams. *J Exp Algorithmics (JEA)* 17:2–4
- Aggarwal CC (2013) A survey of stream clustering algorithms. In: Aggarwal CC, Reddy CK (eds) *Data clustering: algorithms and applications*. Chapman and Hall/CRC, Boca Raton, p 231
- Aggarwal CC, Han J (2014) *Frequent pattern mining*. Springer, Cham
- Aggarwal CC, Han J, Wang J, Yu PS (2003) A framework for clustering evolving data streams. In: Proceedings of the 29th international conference on very large data bases, vol 29. VLDB Endowment, pp 81–92
- Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Bunemann P, Jajodia S (eds) *Proceedings of the 1993 ACM SIGMOD conference on management of data*. ACM Press, New York, pp 207–216

- Alberg D, Last M, Kandel A (2012) Knowledge discovery in data streams with regression tree methods. Wiley Interdiscip Rev Data Min Knowl Disc 2(1): 69–78
- Auer P, Cesa-Bianchi N, Freund Y, Schapire RE (2002) The nonstochastic multiarmed bandit problem. SIAM J Comput 32(1):48–77
- Bach S, Maloof M (2010) A Bayesian approach to concept drift. In: Advances in neural information processing systems. Curran Associates, Inc., New York, pp 127–135
- Bifet A (2010) Adaptive stream mining: Pattern learning and mining from evolving data streams. In: Proceedings of the 2010 conference on adaptive stream mining: pattern learning and mining from evolving data streams. IOS Press, pp 1–212
- Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: International symposium on intelligent data analysis. Springer, pp 249–260
- Bifet A, Read J, Pfahringer B, Holmes G, Žliobaitė I (2013) CD-MOA: change detection framework for massive online analysis. In: International symposium on intelligent data analysis. Springer, pp 92–103
- Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022
- Bradley PS, Fayyad UM, Reina C et al (1998) Scaling clustering algorithms to large databases. In: Proceedings of the 4th international conference on knowledge discovery and data mining, pp 9–15
- Brand M (2002) Incremental singular value decomposition of uncertain data with missing values. In: Computer vision-ECCV 2002, pp 707–720
- Bunch JR, Nielsen CP (1978) Updating the singular value decomposition. Numer Math 31(2):111–129
- Calders T, Dexters N, Gillis JJ, Goethals B (2014) Mining frequent itemsets in a stream. Inf Syst 39:233–255
- Canini K, Shi L, Griffiths T (2009) Online inference of topics with latent dirichlet allocation. In: Proceedings of the twelfth international conference on artificial intelligence and statistics, in PMLR, Clearwater Beach, vol 5, pp 65–72
- Cao F, Estert M, Qian W, Zhou A (2006) Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM international conference on data mining. SIAM, pp 328–339
- Chang JH, Lee WS (2003) Estwin: adaptively monitoring the recent change of frequent itemsets over online data streams. In: Proceedings of the 12th international conference on information and knowledge management. ACM, pp 536–539
- Chang JH, Lee WS (2003) Finding recent frequent itemsets adaptively over online data streams. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 487–492
- Chang JH, Lee WS (2006) Finding frequent itemsets over online data streams. Inf Softw Technol 48(7):606–618
- Charikar M, Chen K, Farach-Colton M (2004) Finding frequent items in data streams. Theor Comput Sci 312(1):3–15
- Chen Y, Tu L (2007) Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 133–142
- Cheng J, Ke Y, Ng W (2008a) Maintaining frequent closed itemsets over a sliding window. J Inf Syst 31(3): 191–215
- Cheng J, Ke Y, Ng W (2008b) A survey on algorithms for mining frequent itemsets over data streams. Knowl Inf Syst 16(1):1–27
- Chi Y, Wang H, Philip SY, Muntz RR (2006) Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl Inf Syst 10(3): 265–294
- Deerwester SC, Dumais ST, Landauer TK, Furnas GW, Harshman RA (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407. www.citeSeer.nj.nec.com/deerwester90indexing.html
- Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80
- Elwell R, Polikar R (2009) Incremental learning in nonstationary environments with controlled forgetting. In: International joint conference on neural networks, IJCNN2009. IEEE, pp 771–778
- Ester M, Kriegel HP, Sander J, Xu X et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of ACM SIGKDD, pp 226–231
- Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. ACM SIGKDD Explorations Newsletter 2(1):51–57
- Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Brazilian symposium on artificial intelligence. Springer, pp 286–295
- Gama J, Rocha R, Medas P (2003) Accurate decision trees for mining high-speed data streams. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 523–528
- Gama J, Rodrigues PP (2007) Stream-based electricity load forecast. In: European conference on principles of data mining and knowledge discovery. Springer, pp 446–453
- Gama J, Rodrigues PP, Lopes L (2011) Clustering distributed sensor data streams using local processing and reduced communication. Intell Data Anal 15(1):3–28
- Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv (CSUR) 46(4):44
- Giannella C, Han J, Pei J, Yan X, Yu PS (2003) Mining frequent patterns in data streams at multiple time granularities. Next Gener Data Mining 212:191–212
- Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng 15(3):515–528
- Günter S, Schraudolph NN, Vishwanathan S (2007) Fast iterative kernel principal component analysis. J Mach Learn Res 8:1893–1918

- Hall P, Marshall D, Martin R (2000) Merging and splitting eigenspace models. *IEEE Trans Pattern Anal Mach Intell* 22(9):1042–1049
- Hartigan JA, Hartigan J (1975) Clustering algorithms, vol 209. Wiley, New York
- Ho Q, Cipar J, Cui H, Lee S, Kim JK, Gibbons PB, Gibson GA, Ganger G, Xing EP (2013) More effective distributed ML via a stale synchronous parallel parameter server. In: Advances in neural information processing systems. Neural Information Processing Systems Foundation, Inc., Lake Tahoe, pp 1223–1231
- Hoffman M, Bach FR, Blei DM (2010) Online learning for latent dirichlet allocation. In: Lafferty JD, Williams CKI, Shawe-Taylor J, Zemel RS, Culotta A (eds) Advances in neural information processing systems. Curran Associates, Inc., New York, pp 856–864
- Honeine P (2012) Online kernel principal component analysis: a reduced-order model. *IEEE Trans Pattern Anal Mach Intell* 34(9):1814–1826
- Ipek E, Mutlu O, Martínez JF, Caruana R (2008) Self-optimizing memory controllers: A reinforcement learning approach. In: Proceedings of 35th international symposium on computer architecture, ISCA’08. IEEE, pp 39–50
- Jagerman R, Eickhoff C, de Rijke M (2017) Computing web-scale topic models using an asynchronous parameter server. In: Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval. ACM
- Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. *ACM Comput Surv (CSUR)* 31(3):264–323
- Jolliffe IT (1986) Principal component analysis and factor analysis. In: Jolliffe IT (ed) Principal component analysis. Springer, New York, pp 115–128
- Kavitha V, Punithavalli M (2010) Clustering time series data stream-a literature survey. arXiv preprint arXiv:1005.4270
- Kim KI, Franz MO, Scholkopf B (2005) Iterative kernel principal component analysis for image modeling. *IEEE Trans Pattern Anal Mach Intell* 27(9):1351–1366
- Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. *Intell Data Anal* 8(3):281–300
- Klinkenberg R, Joachims T (2000) Detecting concept drift with support vector machines. In: ICML, pp 487–494
- Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Proceedings of the 3rd IEEE international conference on data mining, ICDM2003. IEEE, pp 123–130
- Koychev I (2000) Gradual forgetting for adaptation to concept drift. In: Proceedings of the ECAI 2000 workshop on current issues in spatio-temporal reasoning
- Kranen P, Assent I, Baldauf C, Seidl T (2011) The clustree: indexing micro-clusters for anytime stream mining. *Knowl Inf Syst* 29(2):249–272
- Kuncheva LI, Žliobaitė I (2009) On the window size for classification in changing environments. *Intell Data Anal* 13(6):861–872
- Lee D, Lee W (2005) Finding maximal frequent itemsets over online data streams adaptively. In: Proceedings of the 5th IEEE international conference on data mining. IEEE, pp 8–pp
- Leite D, Costa P, Gomide F (2013) Evolving granular neural networks from fuzzy data streams. *Neural Netw* 38:1–16
- Li HF, Ho CC, Lee SY (2009) Incremental updates of closed frequent itemsets over continuous data streams. *Expert Syst Appl* 36(2):2451–2458
- Li HF, Lee SY, Shan MK (2004) An efficient algorithm for mining frequent itemsets over the entire history of data streams. In: Proceedings of the 1st international workshop on knowledge discovery in data streams, vol 39
- Li L, Chu W, Langford J, Schapire RE (2010) A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th international conference on world wide web. ACM, pp 661–670
- Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY (2014) Scaling distributed machine learning with the parameter server. In: Proceedings of 11th USENIX symposium on operating systems design and implementation (OSDI14). USENIX Association, pp 583–598
- Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Mach Learn* 2(4):285–318
- Mahdiraji AR (2009) Clustering data stream: a survey of algorithms. *Int J Knowl Based Intell Eng Syst* 13(2):39–44
- Maloof MA, Michalski RS (2004) Incremental learning with partial instance memory. *Artif Intell* 154(1–2): 95–126
- Minku LL, White AP, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Trans Knowl Data Eng* 22(5): 730–742
- Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533
- Moreno-Torres JG, Raeder T, Alaiz-Rodríguez R, Chawla NV, Herrera F (2012) A unifying view on dataset shift in classification. *Pattern Recog* 45(1):521–530
- O’callaghan L, Mishra N, Meyerson A, Guha S, Motwani R (2002) Streaming-data algorithms for high-quality clustering. In: Proceedings of 18th international conference on data engineering. IEEE, pp 685–694
- Oja E (1982) Simplified neuron model as a principal component analyzer. *J Math Biol* 15(3):267–273
- Oja E (1992) Principal components, minor components, and linear neural networks. *Neural Netw* 5(6):927–935
- Pan SJ, Yang Q (2010) A survey on transfer learning. *IEEE Trans Knowl Data Eng* 22(10):1345–1359
- Pang-Ning T, Steinbach M, Kumar V et al (2006) Introduction to data mining. Pearson Addison Wesley, Boston/Toronto

- Quadrana M, Bifet A, Gavalda R (2015) An efficient closed frequent itemset miner for the MOA stream mining system. *AI Commun* 28(1): 143–158
- Quionero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2009) Dataset shift in machine learning. The MIT Press: Cambridge
- Rodrigues PP, Gama J, Pedroso JP (2006) ODAC: hierarchical clustering of time series data streams. In: Proceedings of the 2006 SIAM international conference on data mining. SIAM, pp 499–503
- Sanger TD (1989) Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Netw* 2(6):459–473
- Schlimmer JC, Granger RH (1986) Incremental learning from noisy data. *Mach Learn* 1(3): 317–354
- Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput* 10(5):1299–1319
- Silva JA, Faria ER, Barros RC, Hruschka ER, de Carvalho AC, Gama J (2013) Data stream clustering: A survey. *ACM Comput Surv (CSUR)* 46(1):13
- Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489
- Smola A, Narayananamurthy S (2010) An architecture for parallel topic models. *Proc VLDB Endow* 3(1–2): 703–710
- Song G, Yang D, Cui B, Zheng B, Liu Y, Xie K (2007) Claim: an efficient method for relaxed frequent closed itemsets mining over stream data. In: International conference on database systems for advanced applications. Springer, pp 664–675
- Song X, Lin CY, Tseng BL, Sun MT (2005) Modeling and predicting personal information dissemination behavior. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery in data mining. ACM, pp 479–488
- Storkey A (2009) When training and test sets are different: characterizing learning transfer. In: Sugiyama C, Lawrence S (eds) Dataset shift in machine learning. MIT Press, Cambridge, pp 3–28
- Sutton RS (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Touretzky DS, Mozer MC, Hasselmo ME (eds) Advances in neural information processing systems, vol 8. MIT Press, Cambridge, pp 1038–1044
- Sutton RS, Barto AG (1998) Reinforcement learning: an introduction, vol 16. MIT Press, Cambridge, pp 285–286
- Syed NA, Liu H, Sung KK (1999) Handling concept drifts in incremental learning with support vector machines. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 317–321
- Teflioudi C, Gemulla R, Myktyuk O (2015) Lemp: fast retrieval of large entries in a matrix product. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, pp 107–122
- Tesauro G (1995) Td-gammon: a self-teaching backgammon program. In: Applications of neural networks. Springer, Boston, pp 267–285
- Tsymbal A (2004) The problem of concept drift: definitions and related work. Technical Report 2, Computer Science Department, Trinity College Dublin
- Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 226–235
- Watkins CJ, Dayan P (1992) Q-learning. *Mach Learn* 8(3–4):279–292
- Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. *Mach Learn* 23(1):69–101
- Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* 8(3–4):229–256
- Xu R, Wunsch D (2008) Clustering, vol 10. Wiley, Hoboken
- Yen SJ, Wu CW, Lee YS, Tseng VS, Hsieh CH (2011) A fast algorithm for mining frequent closed itemsets over stream sliding window. In: 2011 IEEE international conference on fuzzy systems (FUZZ). IEEE, pp 996–1002
- Yu HF, Hsieh CJ, Yun H, Vishwanathan S, Dhillon IS (2015) A scalable asynchronous distributed algorithm for topic modeling. In: Proceedings of the 24th international conference on world wide web, pp 1340–1350. International World Wide Web Conferences Steering Committee
- Yu JX, Chong Z, Lu H, Zhou A (2004) False positive or false negative: mining frequent itemsets from high speed transactional data streams. In: Proceedings of the 13th international conference on very large data bases, vol 30. VLDB Endowment, pp 204–215
- Yuan J, Gao F, Ho Q, Dai W, Wei J, Zheng X, Xing EP, Liu TY, Ma WY (2015) Lightlda: big topic models on modest computer clusters. In: Proceedings of the 24th international conference on world wide web. International World Wide Web Conferences Steering Committee, pp 1351–1361
- Zhang T, Ramakrishnan R, Livny M (1996) Birch: an efficient data clustering method for very large databases. *ACM SIGMOD Rec* 25(2): 103–114
- Zhou A, Cao F, Qian W, Jin C (2008) Tracking clusters in evolving data streams over sliding windows. *Knowl Inf Syst* 15(2):181–214
- Žliabaitė I (2009) Learning under concept drift: an overview. Technical report, Vilnius University
- Žliabaitė I, Bifet A, Gaber M, Gabrys B, Gama J, Minku L, Musial K (2012) Next challenges for adaptive learning systems. *ACM SIGKDD Explor News* 14(1): 48–55

Rendezvous Architectures

Ted Dunning and Ellen Friedman
MapR Technologies and Apache Software
Foundation, Santa Clara, CA, USA

Definitions

Rendezvous architectures are a class of streaming microservice architecture designed to manage multiple implementations of a streaming function so that new implementations can be deployed easily and accurately, input data can be archived precisely, and past operations can be audited while maintaining strict service-level guarantees. Although suitable for more general application, rendezvous architectures are particularly useful for the special case of managing machine learning models. The discussion here follows the trend of associating rendezvous architecture with machine learning and is largely limited to that context.

Historical Background

The management of logistics in machine learning systems has always been notoriously difficult, particularly when there are multiple data sources and, as is almost always the case, multiple models being iteratively developed, evaluated, and deployed at the same time. Requirements include stability and reliability in production along with agility in response to changes. The challenges are even greater when this needs to be done at large scale, in a production setting, and in such a way as to meet critical service-level agreements that increasingly include guaranteed low latency (Böse et al. 2017; Dunning and Friedman 2017). People who are new to machine learning may think that machine learning models can be managed in the same way as normal software, by using continuous integration techniques to deploy a single high-quality model, but the reality for machine learning systems is far different. Successful projects involve a large number of models

both in development and in production simultaneously. Simply determining which model is more accurate is difficult. Moreover, the proliferation of machine learning frameworks means that production models may not share common base technologies. Maintaining strict availability and latency service levels while simultaneously dealing with these other difficulties can be particularly difficult.

Another source of challenges lies in the fact that the process of machine learning model development and evaluation is iterative and needs to operate in a sufficiently flexible and agile way that allows for experimentation and timely response to the need for new or retuned models without compromising operational characteristics like worst-case latency or failure tolerance. Even when models are running well in production, there is still a need for continuous deployment in part because external conditions change. For example, customer behavior may change, fraudsters may develop new tricks, or business goals and requirements may be realigned, thus degrading the performance of a once-effective model.

Additional challenges in machine learning logistics arise from the need for consistency of conditions relative to the production environment during experimentation and evaluation. Even apparently trivial environmental differences between development and production can result in unwanted surprises upon deployment.

It is also important to allow large amounts of experimentation and trials in production settings, but at the same time, it is critical to bound the risk of failed experiments.

Solving these logistical problems with bounded risk is the primary motivation for rendezvous architectures.

Foundations

Rendezvous architectures are best understood in the context of managing machine learning logistics in large-scale systems even though they can be applied to any streaming transformation. In order to meet the challenges of handling

input and output data and managing models, the rendezvous design takes advantage of several key techniques. These include stream-based architecture, a microservice approach, and the use of containers for isolation. The major goals of rendezvous architectures are to be able to:

- Preserve raw data that exactly reflects operational reality.
- Make it easier to manage multiple model versions.
- Provide for smooth and predictable model deployment into production.
- Meet stringent latency and availability guarantees with no planned violations and no accidental violations except under major failure scenarios.
- Work in an iterative and agile fashion.

Overview of Rendezvous Architecture

The distinctive architectural features of rendezvous architectures are the decomposition of a query-response microservice into a set of streaming microservices, the *rendezvous server* itself, and the way that all live models take inputs from and put their results to common streams.

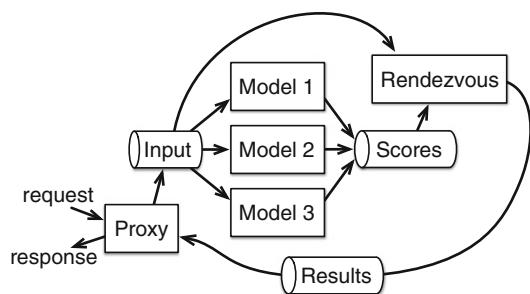
The main rationale for using streaming as the backbone of the rendezvous architecture is that new models can start reading requests from the input stream and writing results into the output stream with no configuration or service discovery burden. This means that the overall management of running models is very simple and dynamic. New model implementations merely need to be told where to get their input and where to put results, and they can start operation. If a model is slow as it warms up or has other problems that prevent it from keeping up with production loads, there is no problem with partial results.

The rendezvous server itself is what makes this work. Results from models for each incoming request are buffered by the rendezvous server until one of the results is received that meets the requirements of a schedule that defines the

trade-off between model preference and latency guarantees. That is, we might have a favored model, but we will only wait for a certain amount of time for that model to give us a result before using a less-favored model. The schedule can even specify a default result to be used in case no model produces a result in the required time.

The overall design of a rendezvous architecture is depicted in Fig. 1. In this figure, a request is accepted by any of a number of proxies and inserted into the `input` stream. All live models run in containers, evaluate as many requests as they can, and put all responses into the `scores` stream. The rendezvous server sees all incoming requests and starts a timer for each such request. As results are received from the live models, they are correlated against pending requests. The rendezvous server has a schedule of preferred model priorities versus latencies, and as soon as a result meets the requirements of the schedule, that result is put into the `results` stream and returned by the original proxy. All other results are ignored by the rendezvous server, although they are preserved in the `scores` stream for a configurable amount of time.

The final step of deploying a new model into production after it has been started and is evaluating requests is for the rendezvous server to stop ignoring the new model's results. This design makes it easy to roll out new models or to roll back to a previous model if performance of the



Rendezvous Architectures, Fig. 1 The general structure of a rendezvous architecture. Incoming requests are handled by one or more proxies that inject requests into a stream so that multiple models can evaluate each request. The rendezvous server sees the original request and selects a result to return to the proxy from the scores stream according to a policy designed to ensure service levels

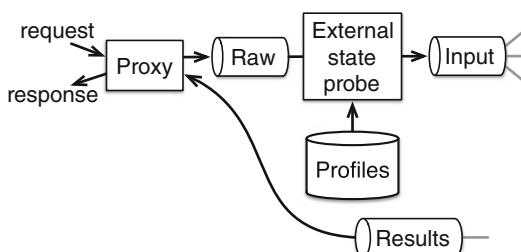
new model is not as expected. Moreover, if the new model crashes or slows down unacceptably, the rendezvous schedule will allow other models to take up the slack and avoid violation of service-level guarantees.

Note that this design also allows raw data to be augmented by additional variables or features before use as input data. This is illustrated in Fig. 2 where the raw request stream is transformed by the addition of common features or insertion of external state information before the models evaluate the requests. By adding common features or injecting external state at this point, all models are guaranteed to have exactly the same inputs.

In order to manage the conflicting environmental requirements for different models, it is a best practice to run each model in a container. This decreases the likelihood that the model behavior will change as it is put into production due to changes in the execution environment. Typically, an orchestration system such as Kubernetes (Burns et al. 2017) is used to manage such containers, but the very simple coordination and discovery for models in a rendezvous architecture make almost any orchestration system acceptable.

Versioning of the Rendezvous Components

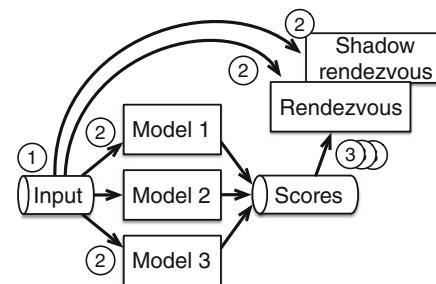
All of the components of the rendezvous architecture can be upgraded during operation with



Rendezvous Architectures, Fig. 2 The incoming requests can be augmented with the result of probing external state (such as user profiles). The same module could be used to compute commonly used input features. This approach avoids race conditions between different models' probes of the external state and makes it easier to maintain exact consistency of all model inputs

no downtime or latency disturbances. This can be done by injecting transition tokens into the input of the system. As these tokens pass through the system, old versions of each component stop processing of messages, and new components take over. For elements such as the feature extraction or external state access which are pure transformations, this is straightforward since all messages are processed entirely by one version or the other. Figure 3 illustrates the more complex case of upgrading the rendezvous server itself. Extra complexity arises because the old rendezvous server has to keep track of any pending requests that arrived before the transition even if the results arrive after the transition, while the new rendezvous server must be sure to keep track of only requests that arrive after the transition token. The process starts with the token at position 1 in the figure at the input to the system. At this point, a shadow rendezvous server will have been started and will be listening to the same inputs as the production server. The token propagates to all models and to the inputs of the current and shadow rendezvous servers at the points marked 2. The models each pass the transition token unchanged so that multiple copies appear in the scores stream.

The production rendezvous server buffers results for all requests that arrive *before* the transition token is seen on the input stream. The



Rendezvous Architectures, Fig. 3 Upgrading the rendezvous server using a transition token that passes through the system to locations marked 1, 2, and 3 at times t_1 , t_2 , and t_3 , respectively. Control over processing transitions in stages from the production rendezvous server to the shadow server that will become the new production server after the transition token has passed all the way through

shadow server takes over the production role by initiating coverage on all requests that are received after the transition token. The overall result is that only one rendezvous server ever produces a result for each request even if a rendezvous server is restarted during the upgrade. The old production rendezvous server can be retired once it reports results for all pending requests. In fact, it may be preferable to keep the old server around for a time in case it becomes necessary to fail back.

Since maximum response latencies for systems where a rendezvous server makes sense are typically less than a second, the hand-off process will appear to be nearly instantaneous once the transition token is introduced to the system.

This upgrade process is related to the Chandy-Lamport (Chandy and Lamport 1985) checkpointing algorithm and to the process that Apache Flink (Friedman and Tzoumas 2016) uses to upgrade program versions but is somewhat simpler.

Key Properties of Rendezvous Architectures

The rendezvous architecture uses streaming microservices (Dunning and Friedman 2016) internally to build a microservice that exposes a query-response interface externally. A rendezvous architecture is characterized primarily by a few properties:

Synchronous requests are evaluated internally in a streaming microservice style. This is highly unusual in large-scale machine learning systems. Much more common is to use a load balancer that distributes individual requests to individual models that evaluate requests synchronously. The use of streams to distributed requests is unusual, and it makes many operations in the rendezvous architecture much easier, largely because persistent streams allow simpler handling of failure modes.

Input requests are sent to all live models identically. In machine learning systems that do not use rendezvous techniques, secondary requests

may be made by the load balancer to fallback models, mostly as hedges against latency violations. Similarly, requests are sometimes broken into pieces and sent to shared decision engines. Results are collected until complete or until a deadline looms and the results are “complete enough.” Doing this with streaming and nearly universally for all queries as in the rendezvous server is very unusual but it deployment of models and the framework itself. It also makes it relatively easy to guarantee response latencies even in the presence of process failures or during the deployment and warmup of new models or the retirement of old ones.

All input requests are evaluated by multiple models. In non-rendezvous systems, evaluation of individual requests by multiple models at the same time, sometimes known as speculative evaluation, is the exception rather than the rule. Historically, high degrees of speculative execution were avoided due to worries about computational capacity, but this is much less of a consideration now, so the aggressive speculative execution of the rendezvous architecture is more viable.

All model results are put into a common stream of results. This is a unique characteristic of rendezvous architectures which allows the implementation of the rendezvous server to be substantially simplified.

A rendezvous server selects which result for each request to return Allowing the rendezvous server to select which result to return according to a trade-off schedule allows the concerns of accuracy and reliability to be separated. Accuracy, the primary goal of the model developer, is embodied in the models themselves. Reliability, particularly with respect to the satisfaction of service-level guarantees, is the focus of site reliability engineers and operations staff and is the key purpose of the rendezvous server. Separating these concerns makes it easier to satisfy both.

Key Applications

The primary application of rendezvous architectures is to allow continuous integration of machine learning models while maintaining promised service levels.

The types of machine learning for which the rendezvous pattern of architecture is most appropriate are those that involve decisioning, that is to say, applications that should return a “correct” answer as estimated by a model without much correlation between different decisions. These systems generally are synchronous in design in that they involve a query-response pattern of interaction with bounded request and response sizes. Examples include predictive analytics, image or speech recognition using deep learning techniques such as medical image analysis or speech-to-text, fraud detection in financial transactions, and IoT sensor data processing for manufacturing or for churn prediction in telecommunications and for web-based decision systems.

Cross-References

- ▶ [Elasticity](#)
- ▶ [Streaming Microservices](#)

References

- Böse JH, Flunkert V, Gasthaus J, Januschowski T, Lange D, Salinas D, Schelter S, Seeger M, Wang Y (2017) Probabilistic demand forecasting at scale. Proc VLDB Endow 10:1694–1705
- Burns B, Hightower K, Beda J (2017) Kubernetes: up and running dive into the future of infrastructure. O’Reilly Media Inc., Sebastopol
- Chandy KM, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM Trans Comput Syst 3(1):63–75. <http://doi.acm.org/10.1145/214451.214456>
- Dunning T, Friedman E (2016) Streaming architecture using Apache Kafka and MapR streams. O’Reilly Media. <http://bit.ly/streaming-dunning>

Dunning T, Friedman E (2017) Machine learning logistics: model management in the real world. O’Reilly, Sebastopol. <https://mapr.com/ebook/machine-learning-logistics/>

Friedman E, Tzoumas K (2016) Introduction to Apache flink: stream processing for real time and beyond. O’Reilly Media Inc. <https://mapr.com/introduction-to-apache-flink/>

Research Directions

- ▶ [Big Data in Computer Network Monitoring](#)

Resource Management

- ▶ [Advancements in YARN Resource Manager](#)

Review

- ▶ [Auditing](#)

Robust Data Partitioning

Alekh Jindal¹, Anil Shanbhag², and Yi Lu²

¹Microsoft, Redmond, WA, USA

²MIT, Cambridge, MA, USA

R

Synonyms

[Adaptive partitioning](#); [Ad Hoc Query Processing](#); [Distributed databases](#)

Definitions

This chapter described new advancements in data partitioning for modern applications that are ad-

hoc in nature and do not have any upfront query workload.

Overview

Data partitioning is a well-known technique for improving the performance of database applications. By splitting data into partitions and only accessing those that are needed to answer a query, databases can avoid reading data that is not relevant to the query being executed, often significantly improving performance. Additionally, when partitions are spread across multiple machines, databases can effectively parallelize query processing across them.

This chapter summarizes traditional data partitioning techniques, motivates the need for a more *robust* data partitioning over modern ad hoc query workloads, introduces the concept of hyper-partitioning for creating a robust partitioning tree and hyper-join to process join queries over such a partitioning tree, and finally discusses repartitioning techniques for adapting the partitioning tree in a robust manner.

Traditional Partitioning Approaches

The traditional approach to data partitioning is to split a table on some key, using hash or range partitioning. This helps queries that have selection predicates involving the key go faster, by only accessing the relevant portions of data. Likewise, for queries with joins, queries will benefit when the database is partitioned on attributes involved in the join, due to local co-partitioned join processing in each partition. Because of these performance gains, many techniques have been proposed in the literature.

Workload-Based Partitioning

The typical approach is to find a good data partitioning for a given query workload. These approaches assume that the query workload is either provided upfront or collected over time, and try to choose the best partitioning for that workload. Examples include fine-grained parti-

tioning (Curino et al. 2010), hybrid of fine- and coarse-grained partitioning (Quamar et al. 2013), skew-aware partitioning (Pavlo et al. 2012), deep integration of partitioning with the query optimizer (Nehme and Bruno 2011), interdependence of different physical design decisions (Zilio et al. 2004), integrating vertical and horizontal partitioning decisions (Agrawal et al. 2004), and partitioning a B⁺-Tree on primary keys (Graefe 2003). Workload-based partitioning need to be reconfigured every time the workload changes.

Multidimensional Partitioning

Several partitioning techniques have been proposed for multidimensional data, e.g., k-d trees, R-trees, and quadtrees. These are typically used for spatial data with two dimensions. Other approaches include binary search trees such as splay trees (Sleator and Tarjan 1985) and MAGIC to *decluster* data on multiple attributes (Ghandeharizadeh and DeWitt 1994). Recent approaches layer multidimensional index structures over distributed data in large clusters. This includes SpatialHadoop (Eldawy and Mokbel 2015), MD-HBase (Nishimura et al. 2011), and epiC (Wang et al. 2010) or adapting the multidimensional index to the workload in TrajStore (Cudré-Mauroux et al. 2010). Commercially, Oracle and MySQL support *sub-partitioning* to create nested partitions on multiple attributes. IBM DB2 supports multidimensional clustering tables to cluster data along multiple dimensions and build block-based indices on them.

Big Data Partitioning

Big data storage systems, such as HDFS, partition datasets based on size. Developers can later create attribute-based partitioning using a variety of data processing tools, e.g., Apache Hive and SCOPE (Zhou et al. 2012). However, such a partitioning is no different than traditional database partitioning since (i) partitioning is a static one time activity and (ii) the partitioning keys must be known a priori and provided by users. Recently, Sun et al. (2014) proposed to create data blocks in HDFS based on the features extracted from each input tuple. Again, the fea-

tures are selected based on a workload, and the goal is to cluster tuples with similar features in the same data block. AQWA looks at adaptive data partitioning for spatial data (two dimensions). Their techniques do not scale to higher dimensions (Aly et al. 2015). Apart from single table partitioning, Hadoop++ (Dittrich et al. 2010) and CoHadoop (Eltabakh et al. 2011) propose to co-partition datasets in HDFS to speed up join queries. These systems still assume a workload.

Database Cracking

Database cracking (Idreos et al. 2007) is a technique to adapt the layout of data and indexes as queries arrive. Partial sideways cracking extends this idea to generate adaptive indexes on multiple columns (Idreos et al. 2009). Cracking is designed for in-memory column stores, and it adapts the data to *every* query in the system. It does not naturally apply to a distributed setting for two main reasons. First, the cost of repartitioning in a distributed setting is higher than in a main memory system. So, it is very expensive to repartition data on every access as cracking does. Second, cracking splits the data on every new predicate it encounters, which can result in a large number of blocks. However, in a distributed setting, the number of data blocks that can be created is limited because blocks must be a certain size to amortize latencies of disk and network access. As a result, adding a split for a new predicate involves merging existing partitions and re-splitting them to keep the number of blocks constant.

Robustness

Modern data analytics has newer data partitioning needs. Data science, for instance, often involves looking for anomalies and trends in data. There is no representative workload for this kind of ad hoc, exploratory analysis, and the set of tables and predicates of interest will often shift over time. For example, an analyst may look for patterns in a database of multidimensional web click events (with user history, demographic informa-

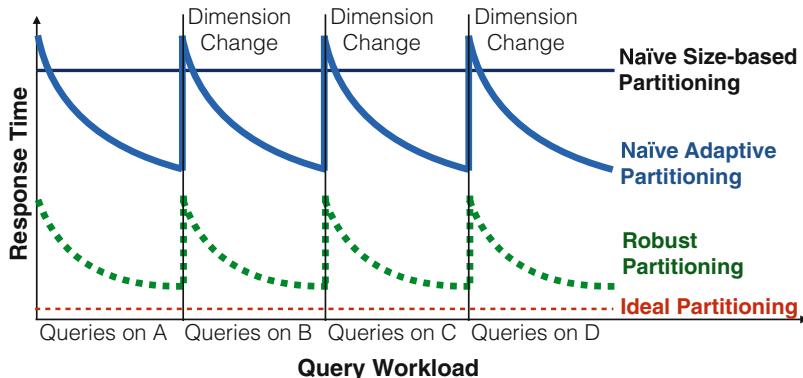
tion, and platform information as dimensions). The analyst may want to view this data according to any of its dimensions – e.g., they may want to query according to the user’s past browsing patterns, by their age or income or by whether they are using a mobile phone or a laptop. As the specific set of attributes of interest is not necessarily known upfront, workload-based partitioning techniques cannot be applied. Furthermore, as the workload is ad hoc in nature, database cracking cannot be applied as well. Figure 1a illustrates the data partitioning dilemma that analysts face with modern workloads.

Analysts are either stuck with naïve size-based partitioning that offers no data skipping capability and hence very poor performance (full scan). Or, alternatively, they could pick one of the more recent adaptive partitioning techniques, e.g., cracking (Idreos et al. 2007) that would make the first few queries even slower than full scan, but will gradually improve if successive queries are on the same dimension, i.e., having a selection predicate on the same attribute. In case the query dimension changes, the performance again goes back worse than full scan before gradually improving with successive queries on the new dimension (referred to as naïve adaptive partitioning). This is really painful for an analyst exploring multiple dimensions: analysts want a data partitioning scheme that is **robust to the ad hoc nature of the modern workloads and provides good performance from the first query itself, adaptively improving from there on**.

R

Hyper-partitioning

Distributed storage systems, such as HDFS, subdivide a dataset into chunks, called blocks, based on size (usually 128 MB). Workload-based partitioning techniques for such systems, including content-based chunking (Bhatotia et al. 2011) and feature-based blocking (Sun et al. 2014), create blocks such that irrelevant blocks could be quickly skipped for the specific query workload. Hyper-partitioning goes a step further by creating blocks based on a partitioning tree that allows to skip data over almost *all* ad hoc queries,



Robust Data Partitioning, Fig. 1 Need for robust data partitioning

without having any information about the query workload. Such a partitioning also serves as a good starting point for an adaptive query executor to improve upon.

Since hyper-partitioning partitions the data along several dimensions, it could end up de-clustering the data blocks across machines and performing random I/Os for each block. However, this is still fine; large block sizes in distributed file systems (Ghemawat et al. 2003) combined with fast network speeds lead to remote reads being almost as fast as local reads (Ananthanarayanan et al. 2011; Binnig et al. 2016). Essentially, hyper-partitioning sacrifices some data locality in order to quickly locate the relevant portions of the data on each machine in a distributed setting.

The rest of this section first introduces the notions of robust partitioning tree and attribute allocations in that tree and then describes how to construct and query such a tree.

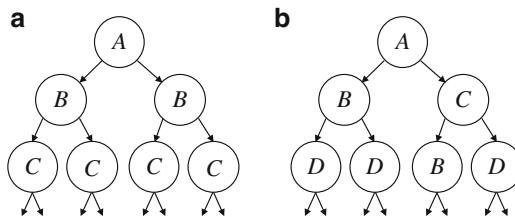
Robust Partitioning Tree

The hyper-partitioning partitioning tree, or simply the *robust tree*, is represented as a balanced binary tree, i.e., the dataset is successively partitioned into two until it reaches the maximum partition size. For HDFS, hyper-partitioning takes the block size as the maximum partition size. The choice of binary tree is deliberate as it is more general (a four-way partitioning can be achieved by two successive two-way partitioning) as well as fine-granular when adapting the tree

to workload changes later. Each node in the tree is represented as A_p , where A is the attribute being partitioned on and p is the cut point. All tuples with $A \leq p$ go to the left subtree and rest go to the right subtree. A leaf node in the tree is a **bucket**, having a unique identifier and a file name in the underlying file system. This file contains the tuples that satisfy the predicates of all nodes traversing upwards from the bucket to the root of the tree. Note that an attribute can appear in multiple nodes in the tree. Having multiple occurrences of an attribute in the same branch of the tree increases the number of ways the data is partitioned on that attribute.

Traditional binary partitioning trees, such as k-d tree (Bentley 1975), partition the space by considering the attributes in a round robin fashion, until the smallest partition size is reached. Hence, the tree can only accommodate as many attributes as the depth of the tree. Figure 2 shows a k-d tree where the three levels of the tree divide the dataset on attributes A , B , and C , respectively. In general, for a dataset size D , minimum partition size P , and n way partitioning over each attribute, the partitioning tree contains $\lfloor \log_n \frac{D}{P} \rfloor$ attributes. With $n = 2$, $D = 1$ TB, and $P = 64$ MB, only 14 attributes can be accommodated in the partitioning tree. However, many real-world schemas have way more attributes.

In contrast to k-d tree, the robust tree performs *heterogeneous branching* in order to accommodate more attributes by partitioning



Robust Data Partitioning, Fig. 2 Multidimensional partitioning tree. (a) k-d tree. (b) Robust tree

different branches of the partitioning tree on different attributes. In other words, robust tree sacrifices the best performance on a few attributes to achieve robustness, i.e., improved performance over more attributes. This is reasonable as without a workload, there is no evident reason to prefer one attribute over another. Figure 2b shows a robust partitioning tree. After partitioning on attribute A , the left side of the tree partitions on B , while the right side partitions on C . Thus, the tree is now able to accommodate *four* attributes, instead of *three*. However, attributes B and D are each partitioned on 75% of the data, while attribute C is partitioned on 50%. Ad hoc queries would now gain partially over all four attributes, which makes the partitioning more effective.

The number of attributes in the robust partitioning tree, with c as the minimum fraction of the data partitioned by each attribute and r as the number of replicas, is given as $\frac{1}{c} \cdot \lceil \log_n \frac{D}{P} \rceil$. With $n = 2$, $D = 1\text{TB}$, $P = 64\text{MB}$, and $c = 50\%$, the number of attributes that can be partitioned is 28. Note that the number of attributes that can be partitioned increases with the dataset size. This shows that with larger dataset sizes, hyper-partitioning is even more useful for quickly finding the relevant portions of the data.

Robust tree can further leverage the data replication in distributed storage systems, e.g., 3 \times replication in HDFS. Such replication mechanisms first partition the dataset into blocks and then replicate each block multiple times. Instead, first, the entire dataset is replicated, and then each replica is partitioned using a different partitioning tree. While the system is still fault tolerant

(because it has the same degree of replication), recovery becomes slower because it needs to read several or all replica blocks in case of a block failure. Essentially, fast recovery time is sacrificed for improved ad hoc query performance. Such a scheme can either increase the number of attributes in the partitioning tree, or increase the data fraction covered per attribute. Both of these lead to improved query performance due to greater partition pruning.

Attribute Allocation

The goal of robust tree is to allocate attributes to nodes in the tree such that all attributes have similar advantage in terms of data skipping or parallel processing. Therefore, the allocation of an attribute is defined as the weighted sum of its fanout on each of the nodes it appears in the partitioning tree T , i.e., the allocation of attribute i is given as:

$$\text{Alloc}_i(T) = \sum_{n \in \text{nodes}(T,i)} \text{DataFraction}_n \cdot \text{Fanout}_n$$

The *allocation* defined above gives the granularity of data partitioning over an attribute. Higher allocation means more data skipping is possible. For example, in Fig. 2b, attribute B appears on two nodes, one covering 50% of the data while the other covering 25% of the data. Thus, B has an allocation of $(0.5 * 2 + 0.25 * 2) = 1.5$. With no query workload, the goal is to balance the benefit of partitioning across all attributes in the dataset. This means that same selectivity predicates on any two attributes X and Y should have similar speedups, compared to scanning the entire dataset. To achieve this, the total allocation is distributed equally among all attributes. Each attribute gets an allocation of $b^{1/|\mathbb{A}|}$, where $|\mathbb{A}|$ is the number of attributes and b is the number of buckets. For instance, if there are eight buckets, and three attributes, the allocation (average fanout) per attribute is $8^{1/3} = 2$. In case of prior workload information, users can provide relative weights of the attributes, and the attribute allocation will be distributed proportional to these weights. The intuition is then to compute the maximum per-attribute allocation and then place

attributes into the tree so as to approximate this ideal allocation.

Hyper-partitioning Algorithm

Algorithm 1 shows the pseudocode to generate the robust partitioning tree. It first calculates the depth of the tree to be created (Line 3), then initializes the queue with the root node of the tree (Line 4), and starts a breadth-first traversal to assign an attribute to every node. The attribute to be assigned at a given node is given by the function `LeastAlloc`, which returns the attribute which has the highest allocation remaining. If two or more attributes have the same highest allocation remaining, the algorithm randomly chooses among the ones that have occurred the least number of times in the path from the node to the root. `Med` returns the median of the attribute assigned to this node by finding the median in the sampled data which comes to this branch. The algorithm starts with an allocation of 2 for the root node, since it partitions the entire dataset into two. Each time it goes to the left or the right subtree, it reduces the data it operates on by half. Once an attribute is assigned to a node,

Algorithm 1: CreateRobustTree

```

Input : Int  $D$ , Int maxPartitionSize, Float[] alloc,
        Tuple[] initSample

1 Tree tree;
2 numBuckets  $\leftarrow \lfloor D/\maxPartitionSize \rfloor$ ;
3 treeDepth  $\leftarrow \log_2(\text{numBuckets})$ ;
4 Queue queue  $\leftarrow \{(\text{tree.root}, \text{treeDepth},$ 
   initSample)\};
5 while queue.size > 0 do
6   node,depth,sample  $\leftarrow$  queue.first();
7   if depth = 0 then
8     node  $\leftarrow$  NewBucket();
9     Continue;
10    node.attr  $\leftarrow$  LeastAlloc(alloc);
11    node.val  $\leftarrow$  Med(sample,node.attr);
12    IS, rS  $\leftarrow$  SplitSample(node.attr,
13      node.val);
14    node.left  $\leftarrow$  CreateNode();
15    node.right  $\leftarrow$  CreateNode();
16    alloc[node.attr]  $\leftarrow 2/2^{\maxDepth - \text{depth}}$ ;
17    depth  $\leftarrow$  1;
18    queue.add((node.left, depth, IS));
19    queue.add((node.right, depth, rS));

```

it subtracts from the overall allocation of the attribute (Line 13). The algorithm creates a leaf-level bucket in case it reaches the maximum depth (Line 18).

Query Processing

A hyper-partitioning query processor considers the filter predicates in incoming queries and filters out partitions that do not match any of the query predicates. For example, if there is a node A_5 in the tree and one of the predicates in the query is $A \leq 4$, then any of the partitions in right subtree of the node don't need to be scanned.

Using Spark, for instance, a job can be constructed where relevant partitions are split into tasks, a set of partitions such that the total size is not more than 4 GB. Each task reads the blocks from HDFS in bulk and iterates over the tuples in main memory. A tuple is returned if it matches the predicates in the query. Tasks are executed independently by the Spark job manager across all machines, and the result is exposed to users as a Spark RDD. Users can use these RDDs to do more analysis using the standard Spark APIs, e.g., run an aggregation.

This section described hyper-partitioning and processing selection queries over a hyper-partitioned input. The following section describes techniques for processing join queries over two or more hyper-partitioned inputs.

Hyper-joins

Hyper-partitioning may end up partially partitioning tables on several different attributes, such that when two tables A and B are joined, a partition in A may join with several partitions in B , each located on HDFS. One option is to simply perform a shuffle join, i.e., repartition both A and B so that each partition of A joins with just one partition of B . However, this can be suboptimal if each partition of A only joins with a few partitions on B ; instead, building a hash table over some partitions of A (or B) and probing it with partitions from B (or A) can result in significantly less network and disk I/O.

Example 1 Suppose table A has three partitions and table B has three partitions. Suppose A_1 joins with B_1 and B_2 ; A_2 joins with B_1 , B_2 , and B_3 ; and A_3 joins with B_2 and B_3 , and each machine \mathcal{M}_i has memory to hold 2 partitions to build hash tables on A . Consider building a hash table over A_1 and A_3 on \mathcal{M}_1 ; we will need to read B_1 , B_2 , and B_3 . We then build another hash table over A_2 on \mathcal{M}_2 and again read B_1 , B_2 , and B_3 . In total, we read six blocks. As an alternative, building a hash table over A_1 and A_2 on \mathcal{M}_1 and another one over A_3 on \mathcal{M}_2 requires reading just B_1 , $2 * B_2$, $2 * B_3 = 5$ blocks.

Thus, building hash tables over different subsets of partitions will result in different costs. Unfortunately, finding the optimal collection of partitions to read is NP-Hard. However, heuristically, solving the problem can still provide significant performance gains over shuffling. To obtain these gains, partitions must be constructed such that, for a join between tables A and B , each partition of A only joins with a subset of the partitions of B . Hyper-join provides this property and is designed to move fewer blocks throughout the cluster than a complete shuffle join when tables are not co-partitioned.

The rest of this section formulates hyper-join as an optimization problem, presents an optimal solution based on mixed integer programming, introduces an approximate algorithm which can run in a much shorter time, discusses hyper-joins for multiple join predicates, and finally shows a two-phase partitioning technique to add join attributes into the robust partitioning tree.

Problem Definition

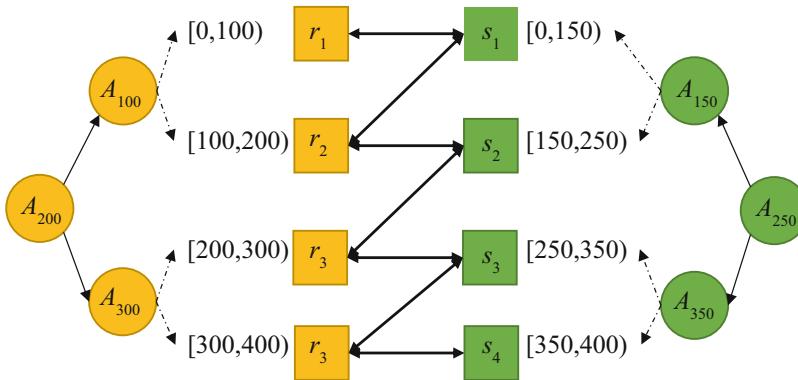
Consider relations R and S , which can join on attribute t . Let $R = \{r_1, r_2, \dots, r_n\}$ and $S = \{s_1, s_2, \dots, s_m\}$ be the collection of data blocks obtained from hyper-partitioning. Let $V = \{v_1, v_2, \dots, v_n\}$ be a collection of m -dimensional vectors, where each vector corresponds to a data block in relation R . The j -th bit of v_i , denoted by v_{ij} , indicates whether block r_i from relation R overlaps with block s_j from relation S on attribute t (these are the

blocks that must be joined with each other). Let $\text{Range}_t(x)$ be a function which gives the range (min and max values) of attribute t in data block x and $\mathbb{1}(s)$ be a function which gives 1 when statement s is true. Given two relations R and S , and for each block r_i from R and s_j from S , let $v_{ij} = \mathbb{1}(\text{Range}_t(r_i) \cap \text{Range}_t(s_j) \neq \emptyset)$. A straightforward algorithm to compute V has a time complexity of $O(nm)$. The Range_t values for each block are stored with each block in the partitioning tree. Let $P = \{p_1, p_2, \dots, p_k\}$ be a partitioning over R , where P is a set of disjoint subsets of the blocks of R and its union is all blocks in R . Each p_i is constrained to be able to fit into memory of the node performing the join. $\tilde{v}(p_i)$ is used to denote the union vector of all vectors in p_i , i.e., $\tilde{v}(p_i) = \bigvee_{r_j \in p_i} v_j$, where v_j is the vector for block r_j . Let $\delta(v_i) = \sum_{k=1}^m v_{ik}$ indicate the number of bits set in v_i . Given a partition p_i , $C(p_i)$ defines the cost of joining p_i with all partitions in S as the number of bits set in $\tilde{v}(p_i)$, i.e., $C(p_i) = \delta(\tilde{v}(p_i))$. This corresponds to the number of blocks to be read to join p_i . Next, the cost function $C(P)$ over a partitioning is defined as the sum of $C(p_i)$ overall p_i in P :

$$C(P) = \sum_{p_i \in P} C(p_i)$$

Thus, the problem of computing hyper join is finding the optimal partitioning P of relation R .

Consider the example in Fig. 3, with table $R = \{r_1, r_2, r_3, r_4\}$ and table $S = \{s_1, s_2, s_3, s_4\}$ and assume $|P| = 2$, i.e., that there is sufficient memory to store $|R|/|P| = 4/2 = 2$ blocks of R in memory at a time. The interval on each partition indicates the minimum and maximum value on the join attribute from all the records. The arrows in the figure indicate the two corresponding partitions overlapping on the join attribute. From the figure, r_1 needs to join with s_1 , r_2 needs to join with s_1 , s_2 , etc. Therefore, $V = \{v_1 = 1000, v_2 = 1100, v_3 = 0110, v_4 = 0011\}$. A hash table could be built over multiple yellow partitions to share some disk access of green partitions. For example, a hash table could be built over the first two yellow blocks (r_1 and r_2) and another one over the last two yellow blocks



Robust Data Partitioning, Fig. 3 Illustrating hyper-join

(r_3 and r_4), so that only 5 green blocks need to be read from disk, assuming only one green block is in memory at a time. In this way, the partition $P = \{p_1 = \{r_1, r_2\}, p_2 = \{r_3, r_4\}\}$, which is optimal. The overall cost $C(P) = 5$, since $\tilde{v}(p_1) = 2$ and $\tilde{v}(p_2) = 3$.

Intuitively, the objective function $C(P)$ is the total number of blocks read from relation S , with some blocks being read multiple times. From the perspective of a real system, the size of p_i is constrained, both due to memory limits and to ensure a minimum degree of parallelism (the number of partitions should be larger than a threshold). If memory is sufficient to hold B blocks from relation R , then we need $c = \lceil n/B \rceil$ partitions. We now define the *minimal partitioning* problem.

Problem 1 Given a set of data blocks from relation R , find a partitioning P over R such that $C(P)$ is minimized, i.e.,

$$\begin{aligned} \arg \min_P \quad & C(P) \\ \text{subject to} \quad & |P| = c, \\ & |p_i| \leq B, \forall p_i \in P. \end{aligned}$$

Optimal Algorithm

This section describes a mixed integer programming formulation which can generate the minimal partitioning. Given the maximum number of data blocks B that can be used to build a hash table due to available worker memory, the total number of hash tables to be built are $c = \lceil n/B \rceil$.

For each data block r_i from relation R and each partition p_k , the assignment of r_i to partition p_k is indicated with a binary decision variable $x_{i,k} \in \{0, 1\}$. Likewise, for each data block s_j from relation S , a binary decision variable $y_{j,k} \in \{0, 1\}$ indicates if the j -th bit of $\tilde{v}(p_k)$ is 1.

The first constraint in Problem 1 requires that the size of each partition p_k is under the memory budget B :

$$\forall k, \quad \sum_{i=1}^n x_{i,k} \leq B$$

The second constraint requires that each data block r_i from relation R is assigned to exactly one partition:

$$\forall i, \quad \sum_{k=1}^c x_{i,k} = 1$$

Given a partitioning P , for each partition p_k , every overlapping data block from relation S must also be in partition p_k . Let J_k be the set of data blocks from relation R which overlaps with data block s_k from relation S .

$$\forall i, \forall k, \forall j \in J_k, \quad y_{i,k} \geq x_{i,j}$$

We seek the minimal input size of relation S :

$$\min \sum_{j=1}^m \sum_{k=1}^c y_{j,k}$$

Solving integer linear programming (ILP) of this form is generally exponential in the number of decision variables; hence the running time of this algorithm may be prohibitive. The proof for NP-hardness of the problem can be found in Lu et al. (2017).

Approximate Solution

Taking B data blocks from relation R with smallest $\delta(\tilde{v}(\mathcal{P}))$ is NP-hard, and there is no algorithm for $n^{1-\epsilon}$ -approximation for any constant $\epsilon > 0$. However, an approximate bottom-up algorithm, as shown in Fig. 4, can provide practical runtimes.

The algorithm starts from an empty set of partitions P and an empty partition \mathcal{P} . It iteratively adds a data block r_i into \mathcal{P} with smallest $\delta(r_i \vee \tilde{v}(\mathcal{P}))$ until there are B blocks in partition \mathcal{P} or no data block left in relation R . It then adds \mathcal{P} into P until P contains all blocks from relation R . A straightforward implementation of this algorithm has a time complexity of $O(n^2)$ (where n is the number of blocks of R), since the minimum cost block (requiring a scan of the non-placed blocks) needs to be computed n times.

Joins Over Multiple Relations

Hyper-join technique can be extended to multiple inputs. Consider TPC-H query 3. If the join order is $(\text{lineitem} \bowtie \text{orders}) \bowtie \text{customer}$ and the intermediate result of the first two tables is denoted by tempLO , then the relation customer needs to join with tempLO on custkey . If custkey is the join attribute in the customer partitioning tree, only tempLO needs to be shuffled based on custkey , and then hyper-join can be used instead of an expensive shuffle join, in which both tempLO and customer need to be shuffled.

Robust Data

Partitioning, Fig. 4 A bottom-up approximate solution

With more relations to join, shuffle join over two intermediate outputs of hyper joins could be more efficient. Consider TPC-H query 8. If the join order is $((\text{lineitem} \bowtie \text{part}) \bowtie \text{orders}) \bowtie \text{customer}$, then the intermediate result with relation lineitem needs to be shuffled twice. Instead, changing the join order to $(\text{lineitem} \bowtie \text{part}) \bowtie (\text{orders} \bowtie \text{customer})$ can use hyper-join twice and a shuffle join over the intermediate results.

Two-Phase Hyper-partitioning

Hyper-join leverages hyper-partitioning; however, the robust partitioning tree described so far partitions data based solely on the selection predicates. Thus, it's unlikely to have the join attribute in very many nodes in the tree, and it's highly possible that every partition will overlap with a large number of partitions. Two-phase partitioning tackles this challenge by injecting the join attributes into the partitioning tree, as depicted in Fig. 5. The first phase splits on join attributes (shown in orange), while the second phase splits on selection attributes (shown in blue). During the first phase, median values of the join attributes are used to recursively split the dataset into two. During the second phase, the join partitions are further partitioned on selection attributes using the standard hyper-partitioning.

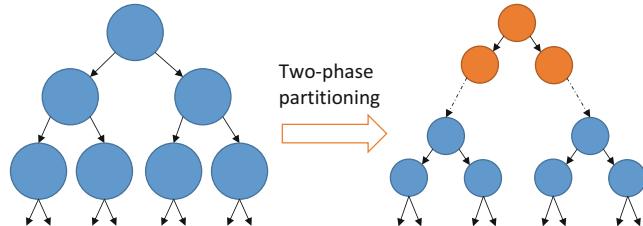
Consider the left partitioning tree in Fig. 3 as an example. There are two levels in the tree which are reserved for the join attribute, which, assuming data is uniformly distributed in the range $[0, 400]$, leads to four disjoint partitions with range $[0, 100)$, $[100, 200)$, $[200, 300)$, and $[300, 400)$. The same procedure is also applied to the right partitioning tree, which creates four disjoint partitions with range $[0, 150)$, $[150, 250)$, $[250, 350)$, and $[350, 400)$.

```

 $R \leftarrow \{r_1, r_2, \dots, r_n\}, P \leftarrow \emptyset, \mathcal{P} \leftarrow \emptyset$ 
while  $R$  is not empty:
    merge  $\mathcal{P}$  with data block  $r_i$  with smallest  $\delta(r_i \vee \tilde{v}(\mathcal{P}))$ 
    if  $|\mathcal{P}| = B$  or  $r_i$  is the last one in  $R$ :
        add  $\mathcal{P}$  to  $P$  and  $\mathcal{P} \leftarrow \emptyset$ 
        remove data block  $r_i$  from  $R$ 
return  $P$ 

```

Robust Data Partitioning, Fig. 5
Illustrating two-phase partitioning



Robust Repartitioning

Hyper-partitioning and hyper-join allow an analyst to quickly get started with her ad hoc queries. However, the analyst also wants the partitioning to adapt as her analysis progresses, e.g., drilling down web click data into successively smaller age groups, to provide even better query performance. *Robust repartitioning* provides the mechanisms to achieve this. When a query is submitted, a repartitioning optimizer explores alternative partitioning trees to find the best one and decides whether repartitioning is worthwhile. The optimized plan only accesses data which is to be read by input queries, i.e., data that is not read by queries during repartitioning is not accessed. This has two benefits: (i) data that is not touched by any query is never repartitioned and (ii) query processing and repartitioning share scans reducing the cost of repartitioning.

The rest of this section describes the cost model used, introduces three basic transformations used to transform a given partitioning tree, describes a divide-and-conquer approach to consider all possible alternatives generated from the transformation rules for inserting a single predicate, discusses how to handle multi-predicate queries, and lastly shows a smooth repartitioning technique to adapt to changing join predicates. It is worth noting that the entire optimization process is transparent to users, i.e., users do not have to worry about making repartitioning decisions and their queries remain unchanged with the new access methods.

Cost Model

Consider a window (W) of queries that happened in the past X hours. X is a parameter in

adaptive query executor, and it determines how quickly the system reacts to workload changes. For each query q in the query sequence, the cost of processing q using partitioning tree T is given as:

$$\text{Cost}(T, q) = \sum_{b \in \text{lookup}(T, q)} n_b$$

where $\text{lookup}(T, q)$ returns the set of relevant buckets for query q in T and n_b is the number of tuples in bucket b . The cost of the query window is the sum of the cost of individual queries. For a query being executed, the optimizer might want to transform the partitioning tree to a new partitioning tree T' resulting in a set of buckets $B \subset \text{lookup}(T, q)$ being repartitioned. The benefit of this transformation is

$$\text{Benefit}(T') = \sum_{q \in W} \text{Cost}(T, q) - \sum_{q \in W} \text{Cost}(T', q)$$

and the added cost of repartitioning is given as

$$\text{RepartitioningCost}(T, q) = c \sum_{b \in B} n_b$$

where c is the write multiplier, i.e., how expensive writes are compared to a read. Repartitioning is expensive; however, it only happens when the resulting decrease in the cost of the query window (benefit) is greater than the repartitioning cost. This check prevents constant re-partitioning due to a random query sequence and bounds the worst case impact. To illustrate, consider a single node in the tree and a query sequence of the form $\sigma_A < 2, \sigma_B < 2, \sigma_A < 2, \sigma_B < 2, \dots$. In this case, the data is not constantly repartitioned. After doing it once, say on A , the total cost

goes down, and hence the repartitioning on B would not happen as Benefit < Repartitioning-Cost.

Tree Transformations

A set of transformation rules allow exploring the space of possible plans when repartitioning the data. Consider a query predicate of the form $A \leq p$, denoted as A_p . Only partitioning transformations that are local, i.e., that do not involve rewriting the entire tree, are considered. These local transformations are cheaper and amortize the repartitioning effort over several queries. The three basic transformations are discussed below.

(1) Swap. Replaces an existing node in the partitioning with the incoming query predicate A_p . As only the accessed data is repartitioned, we consider swapping only those nodes whose left *and* right children are fully accessed by the incoming query. Applying swap on an existing node involves reading both sub-branches, and restructuring all partitions beneath the left subtree to contain data satisfying A_p and the right subtree to contain data that does not satisfy A_p . Swaps can happen between different attributes (Fig. 6a), in which case both branches are completely rewritten in the new tree. Swaps can also happen between two predicates of the same attribute (Fig. 6b), in which case the data moves from one branch to the other. For example, in the Fig. 6b, if node $A_{p'}$ is A_{10} and predicate A_p is $A \leq 5$, then data moves from the left branch to the right branch, i.e., the left branch is completely rewritten while the right branch just has new data appended.

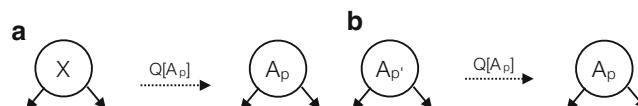
Swaps serve the dual purpose of unpartitioning an existing (less accessed) attribute while refining on another (more accessed) attribute. As both the swap attributes as well as their predicates are driven by the incoming queries, they reduce the access times for the

incoming query predicates. Finally, note that it is cheaper to apply swaps at lower levels in the partitioning tree because less data is rewritten. Applying them at higher levels results in a much higher cost.

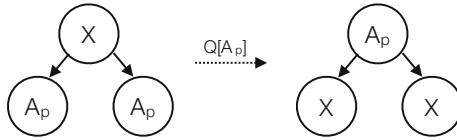
(2) Push-up. Pushes a predicate as high up the tree as possible. This can be done when both the left and the right child of a node contain the incoming predicate, as a result of a previous swap, as shown in Fig. 7. This is a logical partitioning tree transformation, i.e., it only rearranges the internal nodes without any modification to the leaf nodes.

A push-up transformation is checked every time a swap transformation is performed. The idea is to move important predicates (ones that have recently or frequently appeared in the query sequence) progressively up the partitioning tree, from the leaves right up to the root. This makes important predicates less likely to be swapped immediately, because swapping a node higher in the partitioning tree is much more expensive. Another advantage of push-up is that it causes a churn of the attributes assigned to higher nodes in the upfront partitioning. When such a dormant node is pushed down, subsequent predicates can swap them in an incremental fashion, affecting fewer branches, thus making the tree transformations more robust.

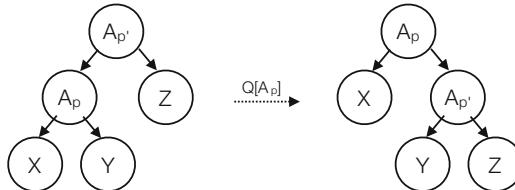
(3) Rotate. Transformation rearranges two predicates on the same attribute such that more important (recently accessed or frequently appearing in the query sequence) predicate appears higher up in the partitioning tree. Figure 8 shows a rotate transformation involving predicates p and p' on attribute A . The goal here is to churn the partitioning tree such that predicates on less important attributes are more likely to be replaced first. Similar to the push-up transformation, rotate is a logical transformation, i.e., it only rearranges the internal nodes of the parti-



Robust Data Partitioning, Fig. 6 Node swap in the partitioning tree **(a)** Different attribute. **(b)** Same attribute



Robust Data Partitioning, Fig. 7 Node pushdown in partitioning tree



Robust Data Partitioning, Fig. 8 Node rotation in partitioning tree

tioning tree and it is always performed wherever possible.

Above three partitioning tree transformations can be combined to capture a fairly general set of repartitioning scenarios. Figure 9 shows an example, where first nodes D_4 are swapped with incoming predicate A_2 at the lower level, then A_2 is pushed up one level above, and finally it is rotated with nodes A_5 and C_3 . In the process, only half the leaves are repartitioned. Thus, in larger trees, repartitioning mostly happens on small fractions of the data modifying a few subtrees locally.

Divide-and-Conquer Repartitioning

Given a query with predicate A_p and a partitioning tree T , there are many different combinations of transformations that need to be considered. However, observe that the data access costs over a subtree T_n , rooted at node n , could be broken down into the access costs over its subtrees, i.e.,

$$\text{Cost}(T_n, q_i) = \text{Cost}(T_{n_{left}}, q_i) + \text{Cost}(T_{n_{right}}, q_i)$$

where $T_{n_{left}}$ and $T_{n_{right}}$ are subtrees rooted respectively at the left and the right child of n . Thus, finding the best partitioning tree can be broken down into recursively finding the best left and right subtrees at each level and considering parent node transformations only on top of the best child subtrees. For each transformation, the

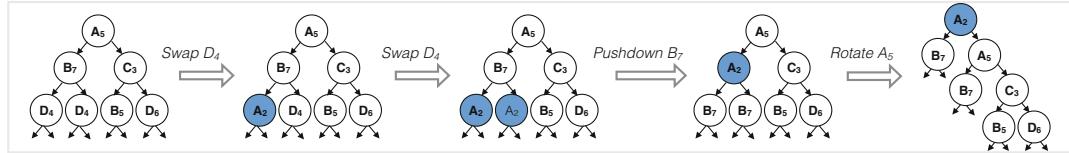
benefit and cost of that transformation is considered, and the one which has the best benefit-to-cost ratio is picked. Table 1 shows the cost and benefit estimates for the different transformations. For the swap transformation, denoted as $P_{\text{swap}}(n, n')$, the query costs are recalculated. However, push-up and rotate transformations, denoted as $P_{\text{swap}}(n, n')$ and $P_{\text{pushup}}(n, n_{\text{left}}, n_{\text{right}})$, respectively, inherit the costs from children subtrees. Applying none of the transformations at a given node is denoted as $P_{\text{none}}(n)$. This approach helps to significantly reduce the candidate set of modified partitioning trees.

Above divide-and-conquer algorithm has a complexity of $O(QN \log N)$, where N is the number of nodes in the tree and Q is the number of queries in the query window. More details on the algorithm can be found in Shanbhag et al. (2017).

Repartitioning with Multiple Predicates

A predicate of the form $A \leq p$ gets inserted in the tree as A_p , and on insertion, only the leaf nodes on the left side of the node are accessed. $A > p$ is also inserted as A_p with the right side of the node being accessed. For $A \geq p$ and $A < p$, let p' be $p - \delta$ where δ is the smallest change for p 's data type. We insert $A_{p'}$ into the tree. $A = p$ is treated as combination of $A \leq p$ and $A > p'$.

Now consider a query with two predicates A_p and A_{p2} . The brute force approach is to consider choosing a set of accessed nonterminal nodes to be replaced by A_p , and then for every such choice, choose a set of remaining nodes to be replaced by A_{p2} . Thus, the number of choices grows exponentially with the number of predicates. A greedy approach is to try to insert each predicate in the query into the partitioning tree. The best among the best plans obtained for different predicates is picked, and the corresponding predicate is removed from the predicate set. Likewise, the remaining predicates are inserted into the best plan obtained so far. The algorithm stops when either all predicates have been inserted or when the tree stops changing. Doing this adds a multiplicative complexity of $O(|P|^2)$ where P is the set of query predicates.



Robust Data Partitioning, Fig. 9 Introducing predicate A_2 into the partitioning tree

Robust Data Partitioning, Table 1 The cost and benefit estimates for different partitioning tree transformations

Transformation	Notation	Cost (C)	Benefit (B)
Swap	$P_{\text{swap}}(n, n')$	$\sum_{b \in T_n} c \cdot n_b$	$\sum_{i=0}^k [\text{Cost}(T_n, q_i) - \text{Cost}(T_{n'}, q_i)]$
Pushup	$P_{\text{pushup}}(n, n_{\text{left}}, n_{\text{right}})$	$C(P(n_{\text{left}})) + C(P(n_{\text{right}}))$	$B(P(n_{\text{left}})) + B(P(n_{\text{right}}))$
Rotate	$P_{\text{rotate}}(p, p')$	$C(P(n_{\text{left} \mid \text{right}})), \text{ for } p' \text{ on } n_{\text{left} \mid \text{right}}$	$B(P(n_{\text{left} \mid \text{right}})), \text{ for } p' \text{ on } n_{\text{left} \mid \text{right}}$
None	$P_{\text{none}}(n)$	$C(P(n_{\text{left}})) + C(P(n_{\text{right}}))$	$B(P(n_{\text{left}})) + B(P(n_{\text{right}}))$

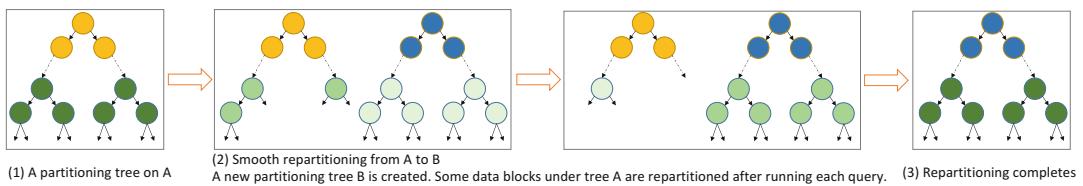
Smooth Repartitioning

A key limitation of the repartitioning techniques presented so far is that they do not adapt in response to join queries. Instead, each table adapts independently, and tables end up being partitioned on different attributes and ranges, such that hyper-join would not provide a performance advantage over shuffle joins. After the initial two-phase partitioning, with new incoming queries containing a new join attribute, the partitioning tree should also shift to the new join attribute. However, repartitioning all of the data immediately would introduce a potentially very long delay and, when the workload is periodic, could lead to oscillatory behavior where it switches from one partitioning to another. Furthermore, a table with multiple foreign keys may join with multiple tables. For example, in TPC-H, queries join lineitem and orders on order_key, and lineitem and supplier join on supplier_key. Smooth repartitioning addresses these challenges by maintaining multiple partitioning trees, building each when a new popular join attribute is seen, and migrating blocks between them. The key goal is to adapt partitioning trees in a way that facilitates joins while still maintaining the performance advantages of partitioning for selection queries.

Smooth partitioning creates a new partitioning (initially empty) tree, when it observes a

query with a new join attribute. The new tree's join attribute comes from the new query, and its predicates are used to build the lower levels of the tree. Smooth repartitioning also repartitions $1/|W|$ of the dataset from the old tree to the new tree, where $|W|$ is the length of the query window. This is accomplished by randomly choosing $1/|W|$ of the blocks in the old tree and inserting them into the new tree (because files are only appended in HDFS, it is possible to do this without affecting the correctness of any concurrent queries). To avoid doing repartitioning work when rare queries arrive, smooth repartitioning can be configured to wait to create a new partitioning tree until the query window contains some minimum frequency f_{\min} of queries for a new join attribute; in this case once the tree is created, $f_{\min}/|W|$ of the blocks will be moved.

As more queries arrive with the new join attribute, smooth repartitioning repartitions more data into the new partitioning tree using the following algorithm. It first calculates the percentage of two types of queries in the query window and the data in each of the partitioning trees. If the incoming query's join attribute is the same as the newly created partitioning tree and the fraction of data in the new partitioning tree is less than the fraction of its type in the query window, data from the old partitioning tree is moved to the new one, again by randomly selecting blocks and moving them.



Robust Data Partitioning, Fig. 10 Illustrating smooth repartitioning

Consider the example in Fig. 10. The algorithm starts from a partitioning tree optimized for join attribute A . When a query with new join attribute B comes, a new partitioning tree for B is created with two-phase partitioning and repartitions $1/|W|$ of the dataset from the old partitioning tree. The color of nodes from the lower levels of the partitioning trees indicates the size of data. The darker the color is, the larger the size of data is. After the new tree is created, both the partitioning trees are maintained with different join attributes. As more queries with join attribute B appear in the query window, more data from the old partitioning tree is repartitioned to the new one. The above procedure is iterated until the query window only includes queries with join attribute B . After the dataset finishes repartitioning, the old partitioning tree for join attribute A is removed, and only the partitioning tree for join attribute B is maintained, which is depicted by the last sub-figure in Fig. 10. (Of course, in many applications, there will not be a complete shift from one join to another, in which case multiple trees will be preserved.)

Conclusion

This chapter described new advancements in data partitioning for modern applications that are ad hoc in nature and do not have any upfront query workload. The key ideas presented include the notion of robustness, the concept of hyper-partitioning for creating a robust partitioning tree without upfront query workload, a hyper-join technique to efficiently process join queries over hyper-partitioned data, and a set of robust repartitioning techniques to steadily adapt the partitioning tree to changes in the workload.

Robust data partitioning revisits the design of a database in the face of modern ad hoc query workloads, recalibrating the database systems to the expectations of modern users – good performance from the first query itself and adaptively improving from there on.

Cross-References

- ▶ [Big Data Indexing](#)
- ▶ [Hadoop](#)
- ▶ [Parallel Join Algorithms in MapReduce](#)

References

- Agrawal S, Narasayya V, Yang B (2004) Integrating vertical and horizontal partitioning into automated physical database design. In: SIGMOD
- Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) AQWA: adaptive query workload aware partitioning of big spatial data. PVLDB 8:2062–2073
- Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2011) Disk-locality in datacenter computing considered irrelevant. In: HotOS
- Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
- Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquin R (2011) Incoop: MapReduce for incremental computations. In: SoCC
- Binnig C, Crotty A, Galakatos A, Kraska T, Zamanian E (2016) The end of slow networks: it's time for a redesign. PVLDB 9:528–539
- Cudré-Mauroux P, Wu E, Madden S (2010) TrajStore: an adaptive storage system for very large trajectory data sets. In: ICDE
- Curino C, Jones E, Zhang Y, Madden S (2010) Schism: a workload-driven approach to database replication and partitioning. PVLDB 3:48–57
- Dittrich J, Quiané-Ruiz JA, Jindal A, Kargin Y, Setty V, Schad J (2010) Hadoop++: making a yellow elephant

- run like a cheetah (without it even noticing). *VLDB* 3:518–529
- Eldawy A, Mokbel MF (2015) Spatialhadoop: a map-reduce framework for spatial data. In: ICDE
- Eltabakh MY, Tian Y, Özcan F, Gemulla R, Krettek A, McPherson J (2011) CoHadoop: flexible data placement and its exploitation in hadoop. *VLDB* 4: 575–585
- Ghandeharizadeh S, DeWitt DJ (1994) MAGIC: a multiattribute declustering mechanism for multiprocessor database machines. *IEEE Trans Parallel Distrib Syst* 5:509–524
- Ghemawat S, Gobioff H, Leung ST (2003) The google file system. *SIGOPS Oper Syst Rev* 37(5):29–43
- Graefe G (2003) Sorting and indexing with partitioned B-trees. In: CIDR
- Idreos S, Kersten M, Manegold S (2007) Database cracking. In: CIDR
- Idreos S, Kersten M, Manegold S (2009) Self-organizing tuple reconstruction in column-stores. In: SIGMOD
- Lu Y, Shanbhag A, Jindal A, Madden S (2017) AdaptDB: adaptive partitioning for distributed joins. *VLDB* 10:589–600
- Nehme R, Bruno N (2011) Automated partitioning design in parallel database systems. In: SIGMOD
- Nishimura S, Das S, Agrawal D, El Abbadi A (2011) MD-HBase: a scalable multi-dimensional data infrastructure for location aware services. In: MDM
- Pavlo A, Curino C, Zdonik S (2012) Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In: SIGMOD
- Quamar A, Kumar KA, Deshpande A (2013) SWORD: scalable workload-aware data placement for transactional workloads. In: EDBT
- Shanbhag A, Jindal A, Madden S, Quiané-Ruiz J, Elmore AJ (2017) A robust partitioning scheme for ad-hoc query workloads. In: SOCC, pp 229–241
- Sleator DD, Tarjan RE (1985) Self-adjusting binary search trees. *J ACM* 32:652–686
- Sun L, Franklin MJ, Krishnan S, Xin RS (2014) Fine-grained partitioning for aggressive data skipping. In: SIGMOD
- Wang J, Wu S, Gao H, Li J, Ooi BC (2010) Indexing multi-dimensional data in a cloud system. In: SIGMOD
- Zhou J, Bruno N, Wu MC, Larson PA, Chaiken R, Shakib D (2012) SCOPE: parallel databases meet MapReduce. *VLDB* 21:611–636
- Zilio DC, Rao J, Lightstone S, Lohman G, Storm A, Garcia-Arellano C, Fadden S (2004) DB2 design advisor: integrated automatic physical database design. In: *VLDB*

Role-Based Access Control (RBAC)

- Security and Privacy in Big Data Environment

Routes

- Spatiotemporal Data: Trajectories

Rule Mining

- Decision Discovery in Business Processes

Rule-Based Processing

- Reasoning at Scale

R

Rule-Oriented Process Mining

- Declarative Process Mining

S

Scala

Alexander Alexandrov
Database Systems and Information Management
Group (DIMA), Technische Universität Berlin,
Berlin, Germany

Definitions

Scala is a statically typed General-purpose Programming Language (GPL) which blends object-oriented and functional programming features. Scala source code compiles to bytecode that runs on the Java Virtual Machine (JVM) and is fully interoperable with code written in Java.

Overview

Scala was designed by Martin Odersky at the École polytechnique fédérale de Lausanne (EPFL) in 2001 and first released in 2003 (Odersky 2006). The original design goal was to combine features from object-oriented and functional programming in a programming language that can be used in practice. The following paragraphs summarize the main features of Scala in comparison to Java.

Objects and Classes

Scala distinguishes between classes – which can be instantiated multiple times using the new

keyword (as in Java) – and objects, which can be understood as a singleton instance of an implicitly derived class definition. An object and a class that share the same name are called companions. Methods and fields that would be declared static in Java must be defined within the companion object of the corresponding class in Scala. For example, the bar method in the following code snippet corresponds to a static method declaration in Java.

```
class Foo {  
    ...  
}  
object Foo {  
    def bar(): Unit = ...  
}
```

Scala also supports Java-like abstract classes using the abstract keyword.

Inheritance and Traits

Instead of interfaces, Scala relies on the concept of traits. Similar to a Java interface, a Scala trait defines methods and fields that can be attached to an object or a class by means of inheritance. In contrast to Java interfaces, however, traits (a) can be partially implemented (which is also possible in Java since version 8) and (b) allow for multiple inheritance – an object or a class can inherit from multiple traits at the same time. The following code snippet defines a class Cat which inherits from Animal, Eyes, and Mouth. Cat instances therefore can call the blink and eat methods defined in their parent traits.

```

abstract class Animal
trait Eyes {
  def blink(): Unit = ...
}
trait Mouth {
  def eat(f: Food): Unit = ...
}
class Cat extends Animal
  with Eyes
  with Mouth

```

Type System

Scala's type system offers more principled support for generics with declaration-site variance, whereas Java allows only for use-site variance. For example, since `Cat` inherits from `Animal`, a covariant type parameter `A` in the generic type `List` (indicated with `+`) allows for assigning instances of type `List[Cat]` to a variable of type `List[Animal]`.

```

class List[+A] { ... }
var xs: List[Animal] = null
xs = List.empty[Cat]

```

In addition, Scala also supports type members and path-dependent types. For example, the following trait defines a trait for comparators for an abstract element type `T`

```

trait Cmp {
  type T
  def cmp(a: T, b: T): Int
}

```

and the `sort` method below accepts a comparator `c` and a sequence of comparable elements of type `c.T` and returns them as a sorted sequence.

```

def sort
  (c: Cmp)
  (s: Seq[c.T]): Seq[c.T]

```

The type `c.T` in the method declaration is *path-dependent* as it depends on the parameter value `c`.

Implicits

Scala implicits encompass a range of language features that enable a number of idiomatic encodings.

Implicit method parameters can be omitted from method applications. The Scala compiler

automatically provides arguments for implicit parameters from the set of implicit values available at the call site. For example, the `sort` method from the previous section can be also declared as follows.

```

def sort[T]
  (s: Seq[T])
  (implicit c: Ord[T]): Seq[T]

```

The above declaration permits incomplete method calls such as

```
sort(Seq(3, 4, 1, 5))
```

if an implicit value of type `Ord[T]` is available in the surrounding lexical scope. A shorthand notation which expands to the above method definition can be used to encode F-bounded polymorphism (Canning et al. 1989) in Scala.

```

def sort[T: Ord]
  (s: Seq[T]): Seq[T]

```

While an unconstrained type parameter `T` corresponds to universal quantification, the `Ord` type constraint corresponds to existential quantification: for all types `T`, the existence of an instance of type `Ord[T]` implies that given a value of type `Seq[T]` we can produce a sorted value of type `Seq[T]`.

TODO

Implicit classes provide a language facility for automatic type conversion. An idiomatic use of this feature is the so-called “pimp my library” pattern. It allows to attach methods and fields to types provided by external libraries in an ad hoc manner. To illustrate the pattern, assume that the `Cat` type from the previous section is provided by an external library that we cannot modify, but we want to add walking facility to `Cat` instances. Implicit classes allow us to achieve that as follows.

```

implicit class CatOps(c: Cat) {
  def walk(): Unit = ...
}
val tom = new Cat
tom.walk()

```

Because the `CatOps` class is declared as `implicit`, the Scala compiler will

automatically wrap the `Cat` instance in the `tom.walk()` call into a `CatOps` constructor call.

```
new CatOps(tom).walk()
```

Implicit conversions are another feature that can be used to encode the “pimp my library” pattern. To that end, instead of declaring the `CatOps` class as `implicit`, we define an `implicit` method called `cat2ops` which converts a `Cat` instance into a `CatOps` instance.

```
implicit def cat2ops(c: Cat) =  
  new CatOps(c)
```

The expanded version of `tom.walk()` now uses the `cat2ops` method instead.

```
cat2ops(tom).walk()
```

Metaprogramming

Metaprogramming is the ability of a programming language to reflect on its own terms and types, manipulate those, and generate new terms and types. Since version 2.10, Scala ships with experimental support for compile-time and run-time metaprogramming in the form of Scala macros (Burmako 2013) and Scala reflection (Coppel et al. 2008). These metaprogramming facilities are extensively used to reduce boilerplate code when designing Domain Specific Languages (DSLs) embedded in Scala, especially in combination with F-bounded polymorphism.

Actor Model and Akka

Akka is a widely used third-party Scala library that implements the actor model for concurrent computation. In the actor model, computation is modeled by actors which exchange messages in an asynchronous manner. Akka hides specifics regarding actor placement behind a uniform API. This allows programmers to run Akka code within same process, within multiple processes on same machine and on different machines without any modifications.

Key Research Findings

From a programming language perspective, the essence of Scala’s type system has been recently formalized in terms of the Dependent Object Types (DOT) calculus and proven sound using a mechanized proof by Rompf and Amin (2016). An influential line of research in Scala-based Embedded Domain Specific Languages (eDSLs) is based on the Lightweight Modular Staging (LMS) framework by Rompf and Odersky (2010). In the domain of Big Data analytics, LMS has been used in the Delite framework (Sujeeth et al. 2014) and a DSL called Jet (Ackermann et al. 2012).

From a data management perspective, Scala has been successfully used as implementation language for a number of research systems for Big Data analytics, most notably Spark (Zaharia et al. 2010) and Stratosphere/Flink (Alexandrov et al. 2014).

Examples of Application

Due to its concise and flexible syntax and its built-in interactive shell, Scala is a popular choice for a host language for eDSLs. In the domain of Big Data analytics, Scala has been popularized by Apache Spark and its Resilient Distributed Dataset (RDD) and Dataset/DataFrame DSLs (Armbrust et al. 2015; Zaharia et al. 2010), by Apache Mahout and its Samsara DSLs (Schelter et al. 2016), and by the Scala DSLs offered by systems such as Summingbird (Boykin et al. 2014) and Apache Flink (Carbone et al. 2015).

Embedded DSLs can be either shallow or deep (Gibbons and Wu 2014). In shallow embedding, eDSL terms evaluate themselves directly at runtime. Contrary, deeply embedded DSLs evaluate themselves in a two steps, first reflecting on their structure in an Intermediate Representation (IR) and then interpreting (and possibly optimizing) this IR. The DSLs mentioned above can be classified as deep, and their embedding mechanism as type-based. The following paragraphs illustrate how the Scala features outlined above are

commonly applied in these DSLs based on their implementation methodology.

Type-based DSLs are structured around a collection of domain-specific types, realized as Scala classes with possible companion objects. The core types in Spark are `RDD` (in the `RDD` DSL) and `Dataset` (in the `Dataset/DataFrame` DSL) which represent a distributed collection managed by the Spark runtime. The core types in Apache Mahout are `Matrix` which represents an in-core matrix and `DrmLike` which represents a Distributed Row Matrix (DRM). The core type in Summingbird is `Producer`, which represents a streaming or batch data producer. Finally, the core types for the various DSLs exposed by Apache Flink are `DataSet`, `DataStream`, or `Table`.

Implicit classes and conversions are often used in the design of type-based DSLs in order to provide ad hoc methods for a specific kind of instances of the associated core DSL types. For example, `RDD`s in Spark offer a set of operators that are only available on an `RDD`s of key-value pairs. The implicit method `rddToPairRDDFunctions` converts value of type `RDD[(K, V)]` into a value of an `RDDOps`-like type `PairRDDFunctions[K, V]`. This allows to leverage the existing Scala infrastructure in order to statically ensure that expressions such as

```
val rdd: RDD[Int, Int] = ...
rdd.countByKey()
```

are valid, while expressions such as

```
val rdd: RDD[Int] = ...
rdd.countByKey()
```

fail with a Scala type error due to a missing implicit conversion.

Embedded DSLs for data-intensive analytics hosted in Scala also rely on Scala's metaprogramming facilities, usually in conjunction with F-bounded polymorphism. For example, in order to reduce the pressure on the JVM garbage collector, the `Dataset` DSL in Spark and the DSLs offered by Flink rely on engine-specific memory management on serialized data. This

means that for a distributed collection of type `Dataset[A]` (in Spark) or `DataSet[A]` (in Flink), the runtime needs to know how to serialize and deserialize instances of the generic element type `A`. To ensure that, every DSL method that changes the element type of the enclosing distributed collection is constrained by an associated type that provides the encoding and decoding functionality for the new element type. For example, the `map` method in Spark is constrained by an `Encoder[B]` instance

```
def map[B: Encoder] (
  f: A => B
): Dataset[B]
```

and the corresponding method in Flink by a `TypeInformation[B]` instance.

```
def map[B: TypeInformation] (
  f: A => B
): Dataset[B]
```

The frameworks also provide generic implementations that can implicitly synthesize `Encoder[A]` and `TypeInformation[A]` instances based on reflected type information about the type argument `A`. Spark's implementation is based on Scala's runtime reflection library, while Flink's implementation is based on Scala macros.

Future Directions for Research

In the area of core language development, current and future research in Scala is mostly related to the next-generation Scala compiler called Dotty developed at the EPFL (Odersky et al. 2016, 2018; Odersky 2006). The Dotty design aims for a combination of faster compilation times and more principled and sound language design based on the DOT calculus.

In the area of DSL development, future research aims to provide a more stable foundation and better tooling for rapid development of optimizing DSLs embedded in Scala. An important milestone in this direction is the Squid metaprogramming framework by Parreaux et al.

(2018). Squid combines the flexibility of dynamic quasiquotes (in the style offered by Lisp) with the typing and scoping guarantees of static quasiquotes (in the style offered by MetaML).

Cross-References

- ▶ Apache Flink
- ▶ Apache Kafka
- ▶ Apache Mahout
- ▶ Apache Spark
- ▶ Spark SQL

References

- Ackermann S, Jovanovic V, Rompf T, Odersky M (2012) Jet: an embedded DSL for high performance big data processing. In: International workshop on end-to-end management of Big Data (BigData 2012), EPFL-CONF-181673
- Alexandrov A, Bergmann R, Ewen S, Freytag J, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V, Naumann F, Peters M, Rheinländer A, Sax MJ, Schelter S, Höger M, Tzoumas K, Warneke D (2014) The stratosphere platform for big data analytics. VLDB J 23(6):939–964. <https://doi.org/10.1007/s00778-014-0357-y>
- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Sellis TK, Davidson SB, Ives ZG (eds) Proceedings of the 2015 ACM SIGMOD international conference on management of data, 31 May–4 June 2015. ACM, Melbourne, pp 1383–1394. <https://doi.org/10.1145/2723372.2742797>
- Boykin PO, Ritchie S, O’Connell I, Lin JJ (2014) Summingbird: a framework for integrating batch and online mapreduce computations. PVLDB 7(13):1441–1451
- Burmako E (2013) Scala macros: let our powers combine! On how rich syntax and static types work with metaprogramming. In: Proceedings of the 4th workshop on scala, SCALA@ECOOP, 2 July 2013. ACM, Montpellier, pp 3:1–3:10. <https://doi.org/10.1145/2489837.2489840>
- Canning PS, Cook WR, Hill WL, Olthoff WG, Mitchell JC (1989) F-bounded polymorphism for object-oriented programming. In: Stoy JE (ed) Proceedings of the fourth international conference on functional programming languages and computer architecture, FPCA, 11–13 Sept 1989. ACM, London, pp 273–280. <https://doi.org/10.1145/99370.99392>
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink™: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38
- Coppel Y, Odersky M, Dubochet G (2008) Reflecting scala. Semester project report, Laboratory for Programming Methods Ecole Polytechnique Federale de Lausanne, Lausanne
- Gibbons J, Wu N (2014) Folding domain-specific languages: deep and shallow embeddings (functional pearl). In: ICFP. ACM, pp 339–347
- Odersky M (2006) A brief history of scala. Blog Post. http://www.artima.com/scalazine/articles/origins_of_scala.html
- Odersky M, Martres G, Petrushko D (2016) Implementing higher-kinded types in dotty. In: Biboudis A, Jonnalaagedda M, Stucki S, Ureche V (eds) Proceedings of the 7th ACM SIGPLAN symposium on scala, SCALA@SPLASH 2016, 30 Oct–4 Nov 2016. ACM, Amsterdam, pp 51–60. <https://doi.org/10.1145/2998392.2998400>
- Odersky M, Blanvillain O, Liu F, Biboudis A, Miller H, Stucki S (2018) Simplicity: foundations and applications of implicit function types. PACMPL 2(POPL):42:1–42:29. <https://doi.org/10.1145/3158130>
- Parreaux L, Voizard A, Shaikhha A, Koch CE (2018) Unifying analytic and statically-typed quasiquotes. PACMPL 2(POPL):13:1–13:33. <https://doi.org/10.1145/3158101>
- Rompf T, Amin N (2016) Type soundness for dependent object types (DOT). In: Visser E, Smaragdakis Y (eds) Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications, OOPSLA 2016, part of SPLASH 2016, 30 Oct–4 Nov 2016. ACM, Amsterdam, pp 624–641. <https://doi.org/10.1145/2983990.2984008>
- Rompf T, Odersky M (2010) Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. In: Visser E, Järvi J (eds) Generative programming and component engineering, proceedings of the ninth international conference on generative programming and component engineering, GPCE 2010, 10–13 Oct 2010. ACM, Eindhoven, pp 127–136. <https://doi.org/10.1145/1868294.1868314>
- Schelter S, Palumbo A, Quinn S, Marthi S, Musselman A (2016) Samsara: declarative machine learning on distributed dataflow systems. In: NIPS workshop ML-Systems
- Sujeeth AK, Brown KJ, Lee H, Rompf T, Chafi H, Odersky M, Olukotun K (2014) Delite: a compiler architecture for performance-oriented embedded domain-specific languages. ACM Trans Embedded Comput Syst 13(4s):134:1–134:25. <https://doi.org/10.1145/2584665>
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Nahum EM, Xu D (eds) 2nd USENIX workshop on hot topics in cloud computing, HotCloud’10, 22 June 2010. USENIX Association, Boston

Scalable Architectures for Big Data Analysis

Peng Sun and Yonggang Wen

School of Computer Science and Engineering,
Nanyang Technological University, Singapore,
Singapore

Overview

The era of big data is upon us. However, traditional data management and analysis systems, which are mainly based on relational database management system (RDBMS), may not be able to handle the ever-growing data volume. Therefore, it is important to design scalable system architectures to efficiently process big data and exploit their value. This chapter discusses various horizontal and vertical scaling big data platforms, focusing on their architectural principle for big data analysis applications, such as machine learning and graph processing. This chapter could aid users to select right system architectures or platforms for their big data applications.

Introduction

This is an era of big data, evidenced by the sheer volume of data from a variety of sources and its growing rate of generation. According to a report from the International Data Corporation (IDC), the global data volume will grow by a factor of 300, from 130 exabytes (1 exabyte = 10^6 terabytes) to 40,000 exabytes, from 2005 to 2020 (Gantz and Reinsel 2007). These data come from everywhere (Hu et al. 2014), such as user-generated contents (e.g., images and videos) posted to social media sites, transaction records, and embedded sensors used to gather information for IoT (Internet of Things) applications.

The concept of big data has been defined as early as 2001. META group (now Gartner) analyst Doug Laney presented a “3Vs” model (Laney 2001) to define challenges and opportunities of data growth, i.e., increasing volume,

velocity, and variety. Although this description was not originally used to define big data, many industries continue to use this “3Vs” model to describe big data 10 years later (Chen et al. 2014). In 2012, Gartner updated the definition of big data as follows: “*Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization*” (Beyer and Laney 2012).

Traditional data management and analysis systems are mainly based on the relational database management system (RDBMS), which could not efficiently handle big data. Specifically, RDBMSs could only support structured data, while big data typically includes masses of semi-structured and unstructured data. Also, the ever-growing data volume could easily extend beyond the ability of average RDBMSs to capture, store, manage, and analyze big data.

To address these challenges and exploit the value of big data, the research community and industry have proposed various scalable big data analysis platforms. Scalability is the ability of a data analysis system to process increasing amounts of data in an appropriate manner. Typically, a big data analytic system should be able to support very large datasets and could be capable of scaling to address the ever-growing size of complex datasets generated in the future (Hu et al. 2014).

This article discusses various big data analysis platforms and focuses on the scalability and the architectural design. Specifically, we categorize existing big data analysis platforms into two types of scaling: horizontal and vertical scaling. For each type of scaling, we illustrate the architectural principle of several representative general-purpose and dedicated big data platforms. Moreover, this article also discusses the advantages and drawbacks of horizontal and vertical scaling to aid users to select the appropriate platforms for specific big data applications or algorithms.

The rest of this article is organized as follows. Section “[Scalability in Big Data Analysis](#)” presents the fundamental concept of scalability for big data analytics. Section [Horizontal Scaling](#)

Platforms introduces various horizontal scaling big data analysis platforms, including general-purpose systems like Hadoop and Spark and dedicated systems for machine learning and graph processing. Section “Vertical Scaling Platforms” describes vertical scaling big data analysis platforms using graphics processing units (GPUs) and field-programmable gate arrays (FPGAs). A brief conclusion with recommendations for future studies is presented in section “Summary”.

Scalability in Big Data Analysis

To capture, manage, and analyze exploding datasets, scaling has become a key technique for big data analysis. A scalable big data analysis platform could adapt to rapid changes in the growth of data. Different platforms incorporate different scaling techniques to support big data analysis.

Horizontal and Vertical Scaling

Popular big data platforms usually use the following two types of scaling to handle big data:

- **Horizontal Scaling:** Horizontal scaling, which is also known as “scaling out,” involves distributing the computation workload across a cluster with multiple commodity servers. With this approach, a big data analysis platform could easily improve its processing capability by adding independent machines. Typically, each machine runs its own operating system and could communicate with other machines via network to exchange intermediate data and controlling information for big data analysis.
- **Vertical Scaling:** Vertical scaling, which is also known as “scaling up,” involves adding more resources, such as processors and memory, to a single server for improving its data processing capability. With this approach, a big data analysis job runs in a single machine with a single operating system. Note that vertical scaling is often done through multi-threading computation and in-process message passing.

Comparison of Horizontal and Vertical Scaling

Table 1 compares the advantages and drawbacks of horizontal and vertical scaling. Most of the existing single-node data analysis tools could take advantage of faster hardware to improve the processing capability. However, it is hard to continuously scale up vertically after a certain limit (Hu et al. 2014). As a comparison, horizontal scaling could continuously increase system performance by installing new machines. Note that network could easily be the performance bottleneck for big data analysis when scaling out. Horizontal scaling big data analysis platforms must take care of this issue to reduce the communication overhead.

The initial investment costs of horizontal scaling big data platforms are relatively less than vertical scaling platforms. Specially, users could build a small commodity cluster to deal with the current big data analysis workload and install more machines to handle exploding datasets generated in the future. As a comparison, when a data analysis system is designed for vertical scaling only, users are locked into certain minimum initial investment costs driven by the hardware, which could handle current and future workload.

Regarding maintenance costs, vertical scaling big data platforms have an advantage than horizontal scaling platforms. Specifically, when users select vertical scaling platforms, they only need to cool a single server. As a compression, when using horizontal scaling platforms with multiple servers and network components, the cooling system may consume a huge amount of energy, which could significantly increase the maintenance costs (Dayarathna et al. 2016).

S

Horizontal Scaling Platforms

Many horizontal scaling platforms have been proposed for big data analysis. Message Passing Interface (MPI), MapReduce, and Spark are thee widely used general-purpose big data platforms. There are also a lot of dedicated platforms that are proposed for machine learning or graph processing.

Scalable Architectures for Big Data Analysis, Table 1 Advantages and drawbacks of horizontal and vertical scaling for big data analysis

Scaling	Advantages	Drawbacks
Horizontal scaling	• Increase processing capability by adding machines	• Network could easily be the performance bottleneck
	• Could avoid the single point of failure problem	• May have high power consumption to cool a cluster
	• Could handle big data with commodity machines	• Time-consuming to configure multiple machines
Vertical scaling	• Existing data analysis tools and softwares could easily take advantage of faster hardware in a single machine	• High financial investment to improve the processing capability of a single-node data analysis system
	• Easy to manage the hardware within a single machine	• Hard to scale up vertically after a certain limit
	• Lower power consumption with just one machine	• May have the single point of failure problem

General-Purpose Big Data Analysis Platforms

MPI

MPI (Gropp et al. 1999) is a standard communication protocol for programming parallel computers using the peer-to-peer architecture. Typically, MPI is used to set up the communication channel between different computation nodes. Thus, each node could exchange data and control information with each other to execute big data analysis applications. MPI is available for many programming languages and has several well-tested and efficient implementations, such as Open MPI (Gabriel et al. 2004), MPICH (Karonis et al. 2003) and MVAPICH (Panda et al. 2013).

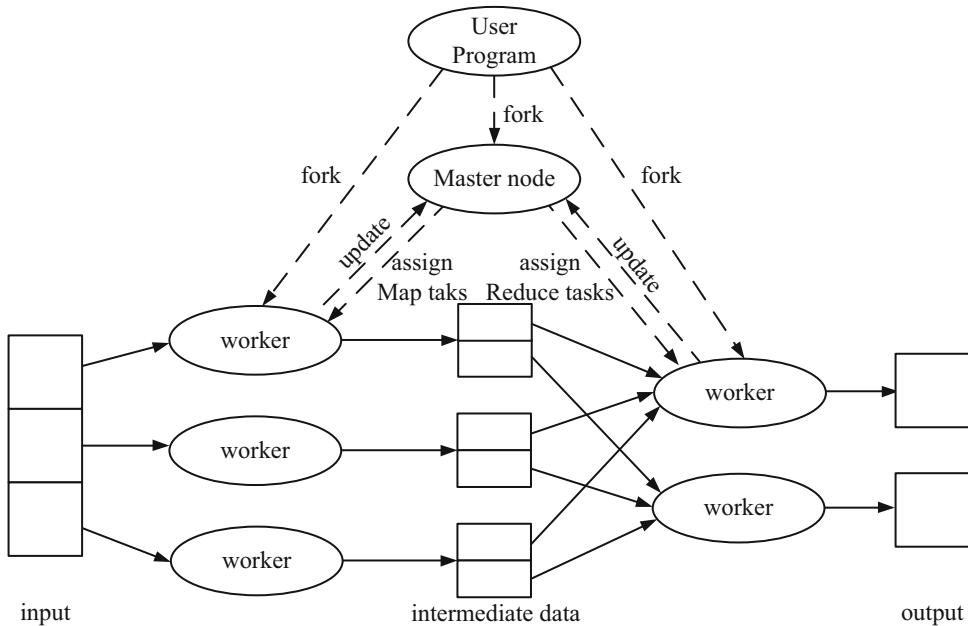
MPI supports both point-to-point and collective communication. More specifically, point-to-point communication is the method to send and receive messages between two individual nodes. Collective communication is a method of communication which involves the participation of all selected nodes. For example, “Broadcast,” one of the standard collective communication techniques, sends the same data to all selected nodes. MPI also provides a set of methods to control the progress of each node. For example, the “Barrier” method puts a barrier and allows all computation nodes to synchronize (reaching up to a certain point) before proceeding further. With these APIs, users could implement their own big data analysis applications and run them in an

MPI-enabled cluster with the specified amount of resources.

Although MPI offers high flexibility for users to develop big data analysis applications, it has some major drawbacks. First, MPI has no mechanism to handle faults. Thus, when running MPI-based big data analysis applications with unreliable hardware (e.g., commodity machines), a single-node failure could cause the whole application to crash. Thus, it is users’ responsibility to design and implement fault-tolerance mechanism when processing big data with MPI. Second, MPI requires more programming changes to design and implement a distributed data analysis application. Hadoop MapReduce and Spark have been designed to addresses these challenges and become popular for big data analysis.

Hadoop MapReduce

Apache Hadoop (White 2012) is an open source project for storing, managing, and processing large-scale datasets using clusters of commodity machines. Hadoop is designed to scale out to hundreds or even thousands of computation nodes. Hadoop contains various components, including Hadoop Distributed File System (HDFS) for reliable big data storage, Hadoop Yarn for efficient cluster resource management, and Hadoop MapReduce for big data analysis.



Scalable Architectures for Big Data Analysis, Fig. 1 Mapreduce architecture

MapReduce, which was firstly proposed by Google (Dean and Ghemawat 2008), is the programming model used in Hadoop. As shown in Fig. 1, there are two types of nodes in a MapReduce cluster: master node and worker. Specifically, the master node (i.e., JobTracker) is in charge of scheduling the submitted MapReduce jobs: partitioning each job into two types of tasks (i.e., Map tasks and Reduce tasks) and assigning them to available workers for execution. Current Hadoop MapReduce uses slots, which is usually configured by the number of CPU cores, to partition the computation resources of a worker. For example, if one server is configured to have two Map slots and two Reduce slots, it could run two Map tasks and two Reduce tasks in parallel.

To perform big data analysis, users just need to write two functions: Map function and Reduce function. When a job is submitted, the underlying MapReduce execution engine will automatically parallelize, distribute, and execute the job on a cluster and process data with two phases, Map phase and Reduce phase, as shown in Fig. 1. Specifically, during the Map phase, workers read input, generate a number of intermediate data, and send them to corresponding workers. During

the Reduce phase, workers process each group of intermediate data in parallel and generate the final results.

One of the major drawbacks of MapReduce is its inefficiency for executing iterative algorithms. Specifically, when executing iterative algorithms with MapReduce, workers would read the same input data from disks at the beginning of each iteration. After each iteration, the results should be written into disks, which may be loaded into memory again in the following iteration. As a result, disk access could be a major performance bottleneck, which significantly degrades the data processing performance. To address this problem, HaLoop (Bu et al. 2010) and Twister (Ekanayake et al. 2010) extend Hadoop MapReduce and improve the performance of executing iterative algorithms by caching input, output, and intermediate data.

S

Spark

Spark (Zaharia et al. 2012) was developed in 2012 in response to limitations in MapReduce. It offers a programming model similar to MapReduce but extends it with a data-sharing abstraction called resilient distributed datasets (RDDs).

RDDs are fault-tolerant, parallel data structures across multiple machines, which let users persist input, output, and intermediate results in memory (if the data does not fit in memory, RDD would also spill it to disk), control their partitioning to optimize data placement, and manipulate them using a rich set of operators. With the help of RDD, Spark could eliminate the disk I/O overhead of Hadoop MapReduce for running iterative big data analysis applications.

The Spark developers have also proposed the Berkeley Data Analytics Stack (BDAS) as an open source software stack to make sense of big data. In BDAS, Alluxio (formerly Tachyon) is a reliable, memory speed, distributed storage system. It could help to reduce data access overhead for big data analysis platforms. BDAS contains a cluster resource management systems called Mesos (Hindman et al. 2011), which could efficiently share cluster resources with multiple distributed computing systems. Spark is the processing engine of BDAS. BDAS also contains many Spark wrappers to optimize the performance Spark for certain applications. For example, Spark Streaming (Zaharia et al. 2013) is used for large-scale stream processing. GraphX (Gonzalez et al. 2014) is proposed for high-performance distributed graph processing. MLlib (Meng et al. 2016) focuses on distributed machine learning.

Dedicated Big Data Analysis Platforms

Many big data analysis platforms have been proposed for dedicated purposes, such as machine learning and graph processing. Compared to aforementioned general-purpose platforms like Hadoop MapReduce and Spark, these dedicated platforms usually could offer much higher performance for certain applications or algorithms.

Machine Learning

Machine learning (ML) builds models from training data and use them to make predictions on new data. It is used in a wide range of applications, including image recognition, object detection, natural language processing, and recommender systems. Typically, an ML model consists of a

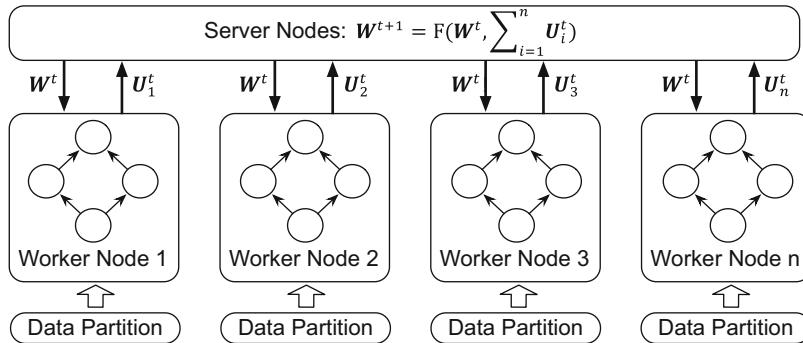
large number of *parameters*, represented as dense or sparse vectors and matrices. To minimize the prediction error, an ML application usually uses an iterative-convergent algorithm to iteratively compute *updates* from training datasets and to aggregate *updates* with the current version of *parameters*.

In industrial ML applications, the size of training data sets could be hundreds to thousands of terabytes. For example, the Yahoo News Feed dataset stands at roughly 110 billion lines of user-news interaction data; ImageNet contains approximately 14 million labeled images; ClueWeb12 has 733 million web pages. Due to the limitation in storage and computation resource, modern single-node ML systems, such as Caffe (Jia et al. 2014) and Theano (Bergstra et al. 2011), cannot cope with such large training datasets.

To handle big training datasets, various distributed ML systems have been proposed based on the parameter server (PS) architecture (Li et al. 2014), such as Petuum (Xing et al. 2015), MXNet (Chen et al. 2015b), Project Adam (Chilimbi et al. 2014), SINGA (Wang et al. 2015), Factorbird (Schelter et al. 2014), and TensorFlow (Abadi et al. 2016). As shown in Fig. 2, the PS architecture can scale to large cluster deployments by having *worker* nodes performing data-parallel computation and having *server* nodes maintaining globally shared *parameters*. When training ML models, *worker* nodes continuously pull latest *parameters* from *server* nodes, perform computation on partitions of the training dataset, and push generated *updates* to *server* nodes, which aggregate them and return a new version of *parameters*. This is done iteratively to bring *parameters* closer to the optimal value.

Graph Processing

Graphs are immensely useful for data mining applications, such as social influence analysis, recommendations, clustering, and anomaly detection. As graph sizes increase exponentially, industrial graph processing applications need to process massive graphs at the scale of millions of vertices and hundreds of billions (or even a trillion) edges (Ching et al. 2015). These massive graphs might not fit in the memory of a single

**Scalable Architectures for Big Data Analysis, Fig. 2**

The PS architecture. In PS, *worker* nodes are in charge of computing *updates*, and *server* nodes manage globally

machine and cannot be processed by existing single-node in-memory graph computation systems, such as Ligra (Shun and Blelloch 2013) and Ligra+ (Shun et al. 2015).

To address single node's memory limit problem, many distributed in-memory graph computation systems have been proposed for fast graph analytics at scale. They usually follow the “think-like-a-vertex” philosophy and abstract graph computation as vertex-centric programs. Specifically, Pregel (Malewicz et al. 2010), Giraph (Ching et al. 2015), Pregel+ (Yan et al. 2014), GPS (Salihoglu and Widom 2013), MOCGraph (Zhou et al. 2014), and HuSky (Yang et al. 2016) adopt the Pregel model: they assign the input graph's vertices to multiple machines and provide interaction between vertices by message passing along out-edges. PowerGraph (Gonzalez et al. 2012), PowerLyra (Chen et al. 2015a), GraphX (Gonzalez et al. 2014), and LiGraph (Zhao et al. 2016) use the Gather-Apply-Scatter (GAS) model: they split a vertex into multiple replicas and could parallelize the computation for a single vertex in different machines. GraphPad (Anderson et al. 2016) and CombBLAS (Buluç et al. 2011) express common graph analyses in generalized sparse matrix-vector multiplication (SpMV) operations and leverage high-performance computing (HPC) techniques to speed up large-scale SpMV. These distributed in-memory approaches, including Pregel-based, GAS-based, and SpMV-based systems, proceed in iterations, where an iteration

shared *parameters*. PS uses a pull/push communication model to exchange data between *worker* nodes and *server* nodes

is called a superstep. In each superstep, several user-based function (UDFs) are called by active vertices to update their values. A lot of benchmarking results (Hu et al. 2014; Iosup et al. 2016) have shown that these dedicated distributed in-memory systems could offer much better performance than general-purpose cluster computing systems like Hadoop MapReduce and Spark.

To further reduce memory footprint of distributed graph processing, researchers have proposed several out-of-core systems to enable large-scale graph processing beyond the memory limit, such as GraphD (Yan et al. 2017), Pregelix (Bu et al. 2014), and Chaos (Roy et al. 2015) which could handle big graph analytics in a small commodity cluster. Typically, these systems manage all or part of vertices in memory and stream edges from disks to reduce memory footprint.

S

Vertical Scaling Platforms

This section introduces vertical scaling platforms utilizing heterogeneous devices: GPU and FPGA.

GPU

GPUs started out as graphics accelerators. In recent years, due to their massively parallel architecture, researchers started to use graphics adapters for non-graphics applications. In this context, many programming frameworks start to give rise to GPGPU (general-purpose computing

on graphics processing units). For example, in 2006, NVIDIA released the first version of CUDA (Compute Unified Device Architecture), a parallel computing platform and programming model for general-purpose algorithms on GPUs.

Take NVIDIA's Pascal architecture as an example, GPUs are connected to RAM via PCIe. A GPU contains multiple streaming multiprocessors (SM). Each SM consists of several single-precision arithmetic logic units (ALUs) ("CUDA Cores"). Therefore, compared to multi-core CPU, GPU usually contains a large number of processing cores. For example, TITAN X, featuring the NVIDIA Pascal architecture, contains 3584 CUDA cores. In addition, GPU also has its own high-throughput DDR5 memory, which is much faster than typical DDR3 memory. Thus, GPUs could offer much higher performance than typical CPUs when executing certain big data analysis algorithms.

Nowadays, GPU is playing an important role in big data analysis, especially when executing machine learning and graph processing algorithms. In a single server, GPU could help Caffe (Jia et al. 2014) and Theano (Bergstra et al. 2011) to speed up the training process of deep learning models by a factor of up to 100. Gunrock (Wang et al. 2016), Medusa (Zhong and He 2014), and CuSha (Khorasani et al. 2014) enable fast and simple graph processing on GPUs.

FPGA

FPGAs contain arrays of programmable logic blocks and a hierarchy of reconfigurable interconnects, which allow the blocks to be "wired together." The FPGA configuration is specified using a hardware description language, such as VHDL or Verilog. Due to customized hardware, FPGAs can be highly optimized for certain big data analysis applications. One of the drawbacks of FPGAs is the high development cost, since FPGAs require developers to have a low-level knowledge of the hardware.

FPGAs are used in various big data analysis applications. For example, FPGAs now are widely used to accelerate deep convolutional neural networks (Zhang et al. 2015; Qiu et al. 2016; Ovtcharov et al. 2015). Graphgen (Nurvitadhi

et al. 2014) and Fpgp (Dai et al. 2016) start to perform vertex-centric graph computation over FPGAs.

Summary

This article surveys various big data analysis platforms and discusses their architectural design and scaling technique. Existing big data platforms could be categorized into two types of scaling: horizontal and vertical scaling. This article illustrates several representative big data analysis platforms for each scaling type and discusses the advantages and drawbacks of these two scaling techniques. This article could aid users to select right platforms for certain applications.

Cross-References

- Scalable Architectures for Big Data Analysis

References

- Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, and Zheng X (2016) Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI'16). USENIX Association, Savannah
- Anderson MJ, Sundaram N, Satish N, Patwary MMA, Willke TL, and Dubey P (2016) Graphpad: optimized graph primitives for parallel and distributed platforms. In: IPDPS. IEEE, pp 313–322
- Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Dafalneau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A et al (2011) Theano: deep learning on GPUs with python. In: NIPS 2011, BigLearning workshop, Granada
- Beyer MA, Laney D (2012) The importance of big data: a definition. Gartner, Stamford, pp 2014–2018
- Bu Y, Howe B, Balazinska M, Ernst MD (2010) Haloop: efficient iterative data processing on large clusters. Proc VLDB Endow 3(1–2):285–296
- Bu Y, Borkar V, Jia J, Carey MJ, Condie T (2014) Pregelix: big(ger) graph analytics on a dataflow engine. Proc VLDB Endow 8(2):161–172

- Buluç A, Gilbert JR (2011) The combinatorial blas: design, implementation, and applications. *Int J High Perfor Comput Appl* 25(4):496–509
- Chen M, Mao S, Liu Y (2014) Big data: a survey. *Mob Netw Appl* 19(2):171–209
- Chen R, Shi J, Chen Y, Chen H (2015a) Powerlyra: differentiated graph computation and partitioning on skewed graphs. In: Proceedings of the tenth European conference on computer systems. ACM, p 1
- Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z (2015b) Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274
- Chilimbi T, Suzue Y, Apacible J, Kalyanaraman K (2014) Project Adam: building an efficient and scalable deep learning training system. In: 11th USENIX symposium on operating systems design and implementation (OSDI'14), pp 571–582
- Ching A, Edunov S, Kabiljo M, Logothetis D, Muthukrishnan S (2015) One trillion edges: graph processing at facebook-scale. *Proc VLDB Endow* 8(12):1804–1815
- Dai G, Chi Y, Wang Y, Yang H (2016) FPGP: graph processing framework on FPGA a case study of breadth-first search. In: Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 105–110
- Dayarathna M, Wen Y, Fan R (2016) Data center energy consumption modeling: a survey. *IEEE Commun Surv Tutorials* 18(1):732–794
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Ekanayake J, Li H, Zhang B, Gunaratne T, Bae SH, Qiu J, Fox G (2010) Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM international symposium on high performance distributed computing. ACM, pp 810–818
- Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A et al (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. In: European parallel virtual machine/message passing interface users group meeting. Springer, pp 97–104
- Gantz J, Reinsel D (2012) The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. *IDC iView IDC Analyze Future* 2007(2012):1–16
- Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) Powergraph: distributed graph-parallel computation on natural graphs. *OSDI* 12(1):2
- Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. In: 11th USENIX symposium on operating systems design and implementation (OSDI'14). USENIX Association, Broomfield, pp 599–613
- Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, vol 1. MIT press, Cambridge
- Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. *NSDI* 11:22–22
- Hu H, Wen Y, Chua TS, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. *IEEE Access* 2:652–687
- Iosup A, Hegeman T, Ngai WL, Heldens S, Prat-Pérez A, Manhardtto T, Chafio H, Capotă M, Sundaram N, Anderson M et al (2016) Ldbc graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc VLDB Endow* 9(13):1317–1328
- Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the ACM international conference on multimedia. ACM, pp 675–678
- Karonis NT, Toonen B, Foster I (2003) Mpich-g2: a grid-enabled implementation of the message passing interface. *J Parallel Distrib Comput* 63(5): 551–563
- Khorasani F, Vora K, Gupta R, Bhuyan LN (2014) Cusha: vertex-centric graph processing on GPUs. In: Proceedings of the 23rd international symposium on High-performance parallel and distributed computing. ACM, pp 239–252
- Laney D (2001) 3D data management: controlling data volume, velocity and variety. *META Group Res Note* 6(70):1–4
- Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY (2014) Scaling distributed machine learning with the parameter server. In: 11th USENIX symposium on operating systems design and implementation (OSDI'14). USENIX Association, Broomfield, pp 583–598
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 135–146
- Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (2011) MLlib: machine learning in apache spark. *J Mach Learn Res* 17(1):1235–1241
- Nurvitadhi E, Weisz G, Wang Y, Hurkat S, Nguyen M, Hoe JC, Martínez JF, Guestrin C (2014) Graphgen: an fpga framework for vertex-centric graph computation. In: IEEE 22nd annual international symposium on field-programmable custom computing machines (FCCM). IEEE, pp 25–28
- Ovtcharov K, Ruwase O, Kim JY, Fowers J, Strauss K, Chung ES (2015) Accelerating deep convolutional neural networks using specialized hardware. Microsoft Res Whitepaper 2(11):1–4
- Panda DK, Tomko K, Schulz K, Majumdar A (2013) The MVAPICH project: evolution and sustainability of an open source production quality MPI library for HPC. In: Workshop on sustainable software for science: practice and experiences, held in conjunction

- with international conference on supercomputing (WSSPE)
- Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, Yu J, Tang T, Xu N, Song S et al (2016) Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 26–35
- Roy A, Bindschaedler L, Malicevic J, Zwaenepoel W (2015) Chaos: scale-out graph processing from secondary storage. In: Proceedings of the 25th symposium on operating systems principles. ACM, pp 410–424
- Salihoglu S, Widom J (2013) GPS: a graph processing system. In: Proceedings of the 25th international conference on scientific and statistical database management. ACM, p 22
- Schelter S, Satuluri V, Zadeh R (2014) Factorbird-a parameter server approach to distributed matrix factorization. arXiv preprint arXiv:1411.0602
- Shun J, Blelloch GE (2013) Ligra: a lightweight graph processing framework for shared memory. In: ACM SIGPLAN notices, vol 48(8). ACM, pp 135–146
- Shun J, Dhulipala L, Blelloch GE (2015) Smaller and faster: parallel processing of compressed graphs with ligra+. In: Data compression conference (DCC). IEEE, pp 403–412
- Wang W, Chen G, Dinh ATT, Gao J, Ooi BC, Tan KL, Wang S (2015) SINGA: putting deep learning in the hands of multimedia users. In: Proceedings of the 23rd ACM international conference on multimedia. ACM, pp 25–34
- Wang Y, Davidson A, Pan Y, Wu Y, Riffel A, Owens JD (2016) Gunrock: a high-performance graph processing library on the GPU. In: ACM SIGPLAN notices 51(8). ACM, p 11
- White T (2012) Hadoop: the definitive guide. O'Reilly Media, Inc., Sebastopol
- Xing EP, Ho Q, Dai W, Kim JK, Wei J, Lee S, Zheng X, Xie P, Kumar A, Yu Y (2015) Petuum: a new platform for distributed machine learning on big data. IEEE Trans Big Data 1(2):49–67
- Yan D, Cheng J, Xing K, Lu Y, Ng W, Bu Y (2014) Pregel algorithms for graph connectivity problems with performance guarantees. Proc VLDB Endow 7(14):1821–1832
- Yan D, Huang Y, Liu M, Chen H, Cheng J, Wu H, Zhang C (2017) Graphd: distributed vertex-centric graph processing beyond the memory limit. IEEE Trans Parallel Distrib Syst 29(1):99–114
- Yang F, Li J, Cheng J (2016) Husky: towards a more efficient and expressive distributed computing framework. Proc VLDB Endow 9(5):420–431
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation. USENIX Association
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM, pp 423–438
- Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J (2015) Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 161–170
- Zhao Y, Yoshigoe K, Bian J, Xie M, Xue Z, Feng Y (2016) A distributed graph-parallel computing system with lightweight communication overhead. IEEE Trans Big Data 2(3):204–218
- Zhong J, He B (2014) Medusa: simplified graph processing on GPUs. IEEE Trans Parallel Distrib Syst 25(6):1543–1552
- Zhou C, Gao J, Sun B, Yu JX (2014) Mocgraph: scalable distributed graph processing using message online computing. Proc VLDB Endow 8(4):377–388
-
- ## Scalable Big Data Privacy with MapReduce
- Sibghat Ullah Bazai¹, Julian Jang-Jaccard¹, and Xuyun Zhang²
- ¹Institute of Natural and Mathematical Sciences, Massey University, Auckland, New Zealand
- ²Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand
- ## Overview
- Processing big data to drive useful information has been in spotlight in recent years. Numerous approaches have been proposed to explore different ways to analyse the big data. However, data privacy has been an issue during the process because data could have been from various sources and they may contain sensitive personal information of individual. Hadoop MapReduce has been considered as one of the most promising approaches for big data processing. This chapter provides an overview of MapReduce environment, privacy challenges faced during the processing of data in MapReduce cluster, existing approaches adopted by various researchers to mitigate these issues. We also provide future guidelines for anonymized data processing to ensure individual privacy in MapReduce.

Introduction

Big data analytics is an emerging technology for finding new insights from large amounts of data. Processing and analyzing these large amounts of data require an extra set of tools and services. Traditional approaches for processing such data unfortunately are no longer effective because of its inherent limitations like computation power, storage capacity, analyzing, visualizing, searching, and sharing (Adnan et al. 2014). For example, Google searching is accessed 38,000 times in 1 s, more than 2.2 billion users send 144 billion emails in 1 day, Facebook manages its users' 40 billion photos, and Walmart stores approximately 2.5 petabytes of data and manages more than million customer transactions per hour (Patel et al. 2012). In the past years, the requirement for processing and storing data used in traditional approaches was quite different than they are today. For example, the traditional approaches did not provide scalability to complete the processing requirement within a given time (Peralta et al. 2015). Scalability is essential when the dataset is too large to be processed on a single machine or the processing becomes unacceptably slow. A careful plan was needed ahead of time to avoid unnecessary hassles, for example, processing the data in a multiple nodes. Similarly, some tasks are simply too time consuming as often computation required to process the data is too complex and resource hungry to run on a single computer. Running machine learning algorithms on large datasets is one of such tasks.

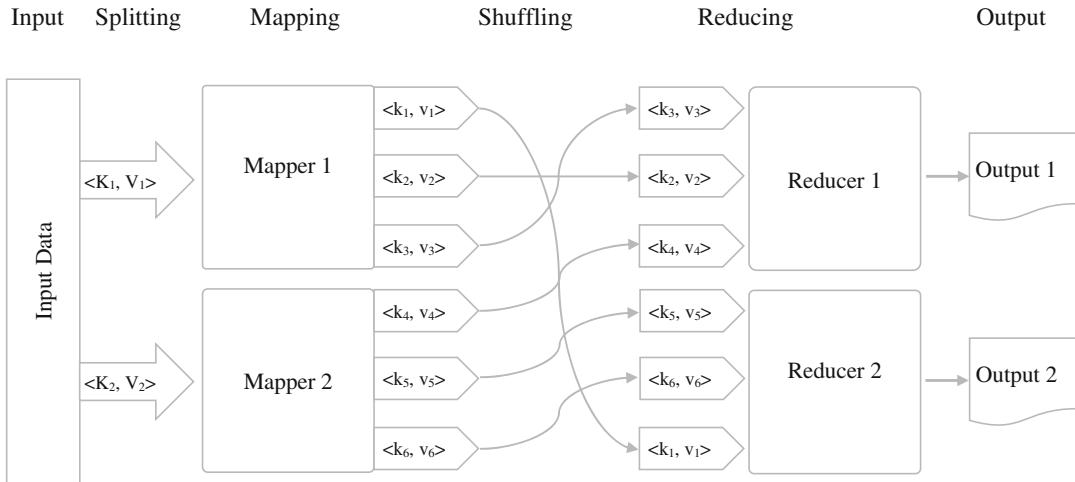
Fortunately, the commodity hardware has become cheaper, and putting several computers together to provide a high processing power has become easier in many cases. Even more, many third-party companies, such as Cloud, offer an hourly rental of hardware/software resources which has made the horizontal scaling very attractive lately. Many frameworks have been designed to use on the commodity hardware/software for specific computing which include MapReduce and Apache Spark among other choices (Dean and Ghemawat 2008; Zaharia et al. 2010).

Data owners are continuing in search of multi-node cluster platform which provides a powerful parallel and distributed architecture to process the big data fast and more efficiently (Aggarwal and Philip 2008). However, the privacy of data processed in such platform has given a data mining community a new concern. There are various techniques to achieve the privacy which includes perturbation, cryptographic-based techniques such as homomorphic encryption and secure multiparty computation, and data anonymization-based techniques such as k-anonymity and differential privacy. However, these techniques exist in isolation from each other in which often are tailored to address a specific problem of a specific domain. This chapter will highlight current big data platform architecture through MapReduce programming model, privacy issues found while processing big data in such architecture, and how existing approaches can (or can't) mitigate the privacy concern. Finally, this chapter will conclude with the future research directions which can provide more advanced approaches to address current privacy concerns.

What Is MapReduce?

The MapReduce-based solutions are considered to be better suited for big data analytics by some as it provides both flexibility and scalability. The MapReduce framework consists of two main functions, mapper and reducer. The mapper function is responsible for splitting an input job (typically large) into a number of smaller manageable jobs. The reducer function collects the divided smaller jobs and processes it further then writes the result to an output file. Figure 1 shows a diagram for the MapReduce programming model.

The mapper function is responsible for splitting an input file into smaller jobs where each job is defined by a key (i.e., an id to a job) and value (i.e., actual job) pairs. The map phase consists of multiple mappers – the same map function is carried out on each mapper. Mappers are executed independently, meaning that there are no interactions among them. There is no



Scalable Big Data Privacy with MapReduce, Fig. 1 The MapReduce programming model

relationship between the input and output types. Figure 1 shows the input to the mapper phase which is abstract as $\langle k_i, V_j \rangle$ and the output of the map phase which is abstract as $\langle k_x, v_y \rangle$. The mapper functions produce partial results, and these partial results are shuffled and sorted by respective key once the mapper function execution is completed. An intermediate data is stored in a local disk which contain the results of shuffled and sorted key value pairs.

Reducer functions collect the intermediate data and perform aggregation on it. Once the mapper tasks are complete, each reducer function accepts a list of values for a same key. The inputs for the reducer function are from the intermediate data. For the list of values, the programmer can define their own process. Finally, the final outputs of the reduce function are emitted at last. The output is still in tuple format (i.e., key and their values) where the key remains the same as input key, while the value is usually the aggregation of all the values in the input. Output tuples are written to their respective files in the file system.

Mapper and reducer functions execute their respective parts separately on different nodes which are often distributed and run in parallel. Hadoop (White 2012) implements MapReduce with Hadoop Distributed File System (HDFS) for an open-source project. This

project automatically handles task distribution, fault tolerance, and other aspects of distributed computing. This makes it much easier for programmers to write parallel processing. It also enables Google to develop a large number of commodity computers to achieve high performance computing at a fraction of the cost compared to a system that is built from a fewer but more expensive high-end servers. MapReduce scales performance by scheduling parallel tasks on several nodes that store the task inputs. Each node executes the tasks with loose communication with other nodes. HDFS is an open-source DFS inspired by Google File System (GFS) and designed to run Hadoop's batch jobs in massive parallelism. HDFS does not implement leases and users can choose which data replica is to be written. HDFS does data-balancing during writes and not periodically like GFS.

Privacy Concern

Unfortunately, distributed data and replicated nature in the MapReduce processing opens up an opportunity for a new huge variety of threats and attacks. While these threats share similarities for attacks in cloud computing models, the effects of attacks for the MapReduce framework are

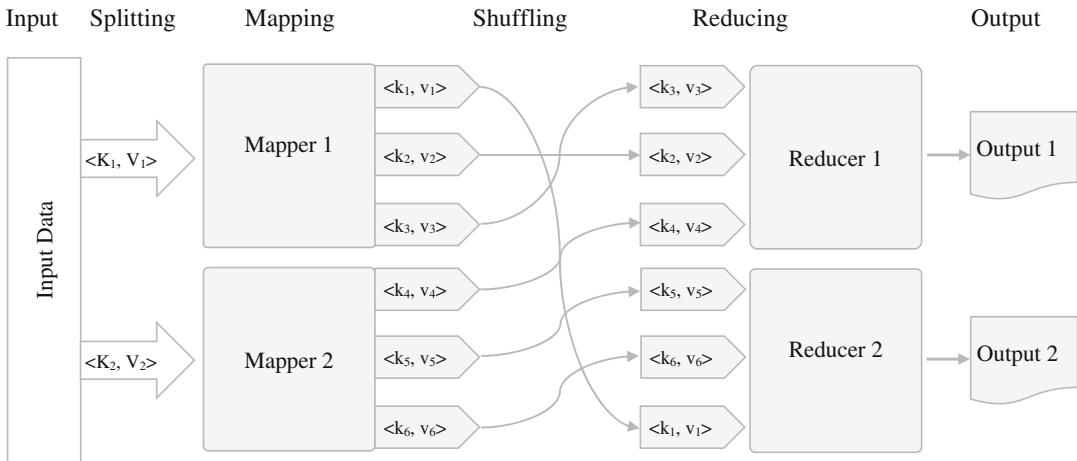
quite different (Derbeko et al. 2016). MapReduce deployed in public clouds is more open to threats and attacks compare when it is deployed in private clouds. The private cloud deployment is more secure because it is deployed in a trusted boundary of a company. This can reduce the danger of attacks and can also prevent company's resources from the threats. Executing the MapReduce jobs in public clouds introduces new challenges in the form of data privacy. In MapReduce, there are multiple mappers and reducers involved in processing data. If one of the many mappers or reducers is compromised by an adversary, it may provide incorrect outputs. A powerful adversary can compromise mapper and reducer function to access information from input, intermediate process, or final output. In these situations, it becomes increasingly difficult to protect individual's privacy in mapper or reducer function. In addition, in MapReduce operations, mappers transform input original key and value pairs to calculate an intermediate key and value pairs, while reducers aggregate an intermediate values set, compute, and write to the output. The output however can bring serious privacy concerns. Firstly, the output can directly leak sensitive information because it contains the global view of the final computation. Secondly, the output can also indirectly leak information via composite attacks where the adversary can link it with public information published via different sources such as Facebook or Twitter. Due to these reasons cloud users hesitate to upload their private information in public clouds. A Cloud computing client may delete sensitive information of individuals present in dataset to ensure their privacy and process the dataset in the public clouds because of its size and convince of computing in the said cloud platform. A Cloud user can delete their sensitive information if the user thinks their data is at risk. However, this is still not enough to protect the user data privacy because third-party administrator controls the public cloud. Third party administrator can easily monitor and eavesdrop client information, and the MapReduce code. Ensuring privacy protection in a compromised cloud service provider is a big concern and can breach the primary

aim of privacy which is to allow clients to use MapReduce services without compromising their data privacy.

Critiques on Emerging Approaches

The conventional anonymization approaches were applied on the output of MapReduce to provide privacy protection. Although it ensures the privacy of data but if the adversary has any access to MapReduce, processing it may leak private information before anonymization is applied to the data. The MapReduce processes data in plain text which is vulnerable from unauthorized monitoring and modification if data operations are executed in untrusted boundaries. In recent years, researchers have adopted anonymization approaches not only on the output but in different stages of MapReduce platform to ensure privacy. In Fig. 2, the areas highlighted in gray are used for anonymization. More number of research is done in the reducer function like Airavat (Roy et al. 2010), Sedic (Zhang et al. 2011), and others. However, the some studies also focus on mapper function to ensure privacy before exaction in plain text and explained as follows.

The current MapReduce framework cannot fully fulfill the requirement for privacy and security features for various applications and infrastructures. One of the cryptographic approaches for privacy-preserving computations is the homomorphic encryption. The technique, considered to be the holy grail of cryptography (Micciancio 2010), allows to perform computation on the cipher text itself without requiring any decryption. The usefulness of such a scheme is evident in the era of cloud computing, particularly in scenarios where organizations are worried about the privacy of their data contents. In traditional cryptographic schemes, the actual data, called plaintext, is encrypted to a distorted representation, called the ciphertext, which is not suitable for direct computations. In such schemes, we must decrypt the ciphertext to the actual data so that we can perform meaningful computations. This, however, is not the case with homomorphic



Scalable Big Data Privacy with MapReduce, Fig. 2 Anonymised MapReduce programming model

encryption schemes where, even after converting to the ciphertext, we can perform certain evaluations without seeing the plaintext. In this way, the plaintext can be kept secret from a third party who can perform calculations on the ciphertext and still provide the correct results. Another most common approach for protecting the security and privacy of data would be to use secure multiparty protocols for computation. This approach has long been studied as in Goldreich et al. (1987). In this technique, a protocol is established between the parties involved such that they all participate in calculating a desired result but without knowing the inputs of other parties (Goldreich 1998). The major cryptographic techniques used in MPC are homomorphic encryption (Cramer et al. 2001) and garbled circuits (Yao 1982). Kamal et al. (Dankar and El Emam 2012) introduce a cryptographic cloud storage service that enables search operation on encrypted data in the cloud, while also ensuring that no information is leaked in the process, and also allows users to verify the integrity of the data at any time. The proposed solution utilizes the combination of attribute-based encryption, searchable encryption, and proof of storage. Mayberry et al. proposed a private information retrieval for MapReduce (PIRMAP). PIRMAP uses mapper phase for parallel computation while homomorphic aggregation is performed in reducer function

(Mayberry et al. 2013). PRISM, a MapReduce technique introduced by Blass et al. (2012), can be efficiently computed in MapReduce. However, these approaches do provide privacy during the execution process but also significantly increase time to encrypt and decrypt the data during computation and do not support many operations on encrypted data.

HybrEx (Ko et al. 2011) and Sedic (Zhang et al. 2011) use MapReduce framework in hybrid cloud to provide privacy for computation data. The HybrEx model presents four execution models for providing privacy to data in a hybrid cloud, that is, map hybrid, horizontal partitioning, vertical partitioning, and hybrid. Each execution model is designed for different setting of map, reducing function execution in multiparty cloud environment. However, the paper does not provide any implementation for any of the proposed model. Sedic implemented the map hybrid model on top of Hadoop, in which reducer operation is only executing in the private cloud without utilizing the public cloud resources. However, the sensitivity of given dataset should be defined in the system before the execution, which eliminates the chances of using chained or iterative MapReduce process. Roy et al. (2010) present Airavat, a security and privacy preservation platform for MapReduce. Airavat ensures individual's data privacy using differential privacy by

adding calculated noise to each query in MapReduce. There are two approaches proposed by the authors. First, it provides mandatory access control (MAC) to ensure only authorized users have access to authorized tasks and resources. The MAC is activated when privacy leakage exceeds a defined limit. Tran and Sato (2012), however, if adversary, manage to sneak reducer code by changing user rights as a trusted user; the proposed solution fails to provide privacy guarantee. Zhang et al. (2012) proposed a privacy-preserving layer over MapReduce, which satisfies privacy demands itemized by data publishers built on diverse MapReduce privacy models. Mohan et al. then present GUPT (Mohan et al. 2012) which is another improved version of Airavat. GUPT defines the differentially private parameters in such a way that it automatically chooses and distributes the privacy parameter. However, Airavat add p reconfigured noise for query which limits its application. Clifton and Tassa (2013) observed that Airavat, unable to pick a suitable privacy budget and sensitivity value for selective quarry, leaves the privacy choice up to the user. In addition, Xiao and Xiao (2014) assign the executing task to new worker node and compare both results to eliminate the malicious node. The existing schemes have certain limitations such as inefficiency to handle incremental data, time taken to update records is more, experiences poor scalability, absence of parallelization, poor execution time, higher rate for loss of information, and inequity between data utility and anonymization.

Future Direction

Processing big data for multi-node cluster involved many vulnerabilities, for example, administrative control of data processing environment, distribution of data on an untrusted node in a public cloud for processing, etc. Victor et al. (2016). To address this limitation, state-of-the-art privacy protection methodologies, such as k-anonymity and differential privacy, have received great attention from the scientific community in recent years, and there are few implementations

offered in the area (Bello-Orgaz et al. 2016). However, these approaches were adopted in the earlier versions of big data processing platform like earlier version of Hadoop. Since the architecture and design for these approaches have been improved radically, the data anonymization algorithm used for previous deployments also needs to be updated to work efficiently (Jain et al. 2016).

K-anonymity ensures the privacy of data where perturbation and cryptographic approaches failed with well-known datasets such as AOL, adult dataset, and Netflix (Bayardo and Agrawal 2005). K-anonymity is the first data anonymization technique with formal mathematical support as a proof. Latanya Sweeney in 2002 (Sweeney 2002) introduced k-anonymity by stating that without ensuring k individuals in aggregation single aggregate statistic should not be published. The formal definition of k-anonymity formalized as each release of the data must be such that every combination of values of quasi-identifiers can be indistinguishably matched to at least k respondents. Quasi-identifiers (QID) are attributes in dataset which may be linked from publicly available dataset by join linkage. The main goal to achieve k-anonymity is to replace QID values with more general values, e.g., generalizing 15, 17, and 19 into a 15–20 value. K-anonymity provides the balance between privacy and utility. It provides more flexibility to data publisher for choosing the desired level of privacy. Most importantly, the anonymized data can be used for processing for analytical data operations even after applying data anonymization technique. Bazai et al. (2017a) proposed an algorithm to use k-anonymity on k-NN classifier for MapReduce operations. The algorithm transformed the plaintext data into anonymized data and executed k-NN classifier on anonymized data. The results provide better data utility for veracious degrees of anonymity. Differential privacy approach has been widely used in statistical queries (Inan et al. 2010). Differential privacy ensures the privacy of individual by answering the query in such a way that the query result does not contain any information about individual's presence in

that data. Differential privacy (Dwork 2008) was formally presented by Dwork in 2006, the idea of adding carefully calibrated noise to the data. The noise level is controlled in such a way that if the records are correlated with other datasets, then an adversary is not able to trace the data subject. The addition of noise in the original data is calibrated so that the noisy data acquired must be usable for data processing in the same way as the original. The noise is not allowed to increase beyond a level where the noisy data becomes so different from the original data that its utilization (accuracy) is impractical. Differential privacy has many advantages over other privacy techniques. Firstly, the mathematical definition of differential privacy is based on two properties: sensitivity and privacy budget. The sensitivity focuses on record closeness to each other, and the privacy budget is calculated based on how much privacy is required for a given query. Secondly, also only unique to this technique, it does concern with adversary computational power and background knowledge (i.e., even if the adversary knows the individual's publicly available details and then uses it for de-anonymization of the query, the adversary cannot de-anonymize it because of random anonymization mechanism used by differential privacy). Thirdly, it maintains composability which ensures that it does not leak privacy even an adversary holds and combines two differentially private results. These properties free data providers from the concern of privacy leakage of their data. Vernica et al. describe KNN processing approach for computing set resemblances on textual documents (Vernica et al. 2010). Their focuses are with document processing rather than business data analytics, and they do not address any implication of privacy; therefore, no data anonymization strategies were mentioned.

In big data processing platforms, computation is done on data in different executions in plaintext; any information leakage while processing the data may disclose sensitive information. Bazai et al. (2017b) mainly focus on addressing this limitation and intended to explore how data anonymization techniques (k-anonymity and dif-

ferential privacy) can be applied in big data processing environment, using MapReduce, which provides a well-known basic set of big data processing architecture and infrastructure, to validate the proposed approach.

Big datasets usually contain many different data types; for example, information is stored in the textual data containing categorical and continuous value, while others are stored in the numerical data. Previous studies show (Fletcher and Islam 2017) that different data types impose different constraints when applying data anonymization techniques. This is true when each privacy parameter has a different impact on the data type. For example, k-group size provides the better utility and privacy with categorical values, while differential privacy works better with well-distributed numerical values (Blum et al. 2013). Many different approaches for using different privacy parameters have been found in the literature for categorical values and numerical data in traditional data processing approaches (Natwichai et al. 2006); however, these implementations do not support multi-node cluster.

Summary

Big data is a very popular term nowadays and has opened a big data era. Distributed and parallel applications are designed to process this huge data to drive new insights and knowledge. Although the design of these applications ensures the data security in a certain level, individual privacy is at a great risk. This chapter illustrates the privacy concern of big data processing platform and provides critiques for existing approaches that are supposed to prevent MapReduce privacy leakage. Although researchers are addressing the privacy concern with various approaches for data analytics in big data processing platforms with perturbation, cryptographic techniques, k-anonymity, and differential privacy, most of the studies are still are in early stages. The practical implementation of anonymity with data analytics will help number of privacy and security challenges.

References

- Adnan M, Afzal M, Aslam M, Jan R, Martinez-Enriquez A (2014) Minimizing big data problems using cloud computing based on hadoop architecture. In: 2014 11th annual high-capacity optical networks and emerging/enabling technologies (HONET). IEEE, pp 99–103
- Aggarwal CC, Philip SY (2008) A general survey of privacy-preserving data mining models and algorithms. In: Privacy-preserving data mining. Springer, Dordrecht, pp 11–52
- Bayardo RJ, Agrawal R (2005) Data privacy through optimal k-anonymization. In: Proceedings of the 21st international conference on data engineering (ICDE 2005). IEEE, pp 217–228
- Bazai SU, Jang-Jaccard J, Wang R (2017a, in press) Anonymizing k-nn classification on mapreduce. In: The 9th EAI international conference on mobile networks and management. Springer
- Bazai SU, Jang-Jaccard J, Zhang X (2017b) A privacy preserving platform for mapreduce. In: International conference on applications and techniques in information security. Springer, pp 88–99
- Bello-Orgaz G, Jung JJ, Camacho D (2016) Social big data: recent achievements and new challenges. Inf Fusion 28:45–59
- Blass EO, Di Pietro R, Molva R, Önen M (2012) Prism-privacy-preserving search in mapreduce. In: Privacy enhancing technologies, vol 7384. Springer, pp 180–200
- Blum A, Ligett K, Roth A (2013) A learning theory approach to noninteractive database privacy. J ACM (JACM) 60(2):12
- Clifton C, Tassa T (2013) On syntactic anonymity and differential privacy. In: 2013 IEEE 29th international conference on data engineering workshops (ICDEW). IEEE, pp 88–93
- Cramer R, Damgård I, Nielsen J (2001) Multiparty computation from threshold homomorphic encryption. In: Advances in cryptology-EUROCRYPT 2001, pp 280–300
- Dankar FK, El Emam K (2012) The application of differential privacy to health data. In: Proceedings of the 2012 joint EDBT/ICDT workshops. ACM, pp 158–166
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
- Derbeko P, Dolev S, Gudes E, Sharma S (2016) Security and privacy aspects in mapreduce on clouds: a survey. Comput Sci Rev 20:1–28
- Dwork C (2008) Differential privacy: a survey of results. In: International conference on theory and applications of models of computation. Springer, pp 1–19
- Fletcher S, Islam MZ (2017) Differentially private random decision forests using smooth sensitivity. Exp Syst Appl 78:16–31
- Goldreich O (1998) Secure multi-party computation. Manuscript preliminary version, pp 86–97
- Goldreich O, Micali S, Wigderson A (1987) How to play any mental game. In: Proceedings of the nineteenth annual ACM symposium on theory of computing. ACM, pp 218–229
- Inan A, Kantarcioğlu M, Ghinita G, Bertino E (2010) Private record matching using differential privacy. In: Proceedings of the 13th international conference on extending database technology. ACM, pp 123–134
- Jain P, Gyanchandani M, Khare N (2016) Big data privacy: a technological perspective and review. J Big Data 3(1):25
- Ko SY, Jeon K, Morales R (2011) The hybrex model for confidentiality and privacy in cloud computing. In: HotCloud, pp 1–8
- Mayberry T, Blass EO, Chan AH (2013) PIRMAP: efficient private information retrieval for mapreduce. In: International conference on financial cryptography and data security. Springer, pp 371–385
- Micciancio D (2010) A first glimpse of cryptography's holy grail. In: Commun ACM 53(3):96–96
- Mohan P, Thakurta A, Shi E, Song D, Culler D (2012) GUPT: privacy preserving data analysis made easy. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. ACM, pp 349–360
- Natwichai J, Li X, Orlowska ME (2006) A reconstruction-based algorithm for classification rules hiding. In: Proceedings of the 17th Australasian database conference, vol 49. Australian Computer Society, Inc., pp 49–58
- Patel AB, Birla M, Nair U (2012) Addressing big data problem using hadoop and map reduce. In: 2012 Nirma University international conference on engineering (NUiCONE). IEEE, pp 1–5
- Peralta D, del Río S, Ramírez-Gallego S, Triguero I, Benítez JM, Herrera F (2015) Evolutionary feature selection for big data classification: a mapreduce approach. In: Math Probl Eng, pp 1–12
- Roy I, Setty ST, Kilzer A, Shmatikov V, Witchel E (2010) Airavat: security and privacy for mapreduce. In: NSDI, vol 10, pp 297–312
- Sweeney L (2002) Achieving k-anonymity privacy protection using generalization and suppression. Int J Uncertain Fuzziness Knowl Based Syst 10(05):571–588
- Tran Q, Sato H (2012) A solution for privacy protection in mapreduce. In: 2012 IEEE 36th annual computer software and applications conference (COMPSAC). IEEE, pp 515–520
- Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using mapreduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 495–506
- Victor N, Lopez D, Abawajy JH (2016) Privacy models for big data: a survey. Int J Big Data Intell 3(1):61–75
- White T (2012) Hadoop: the definitive guide. O'Reilly Media, Inc., Sebastopol
- Xiao Z, Xiao Y (2014) Achieving accountable mapreduce in cloud computing. Futur Gener Comput Syst 30:1–13
- Yao AC (1982) Protocols for secure computations. In: 23rd annual symposium on foundations of computer science, SFCS'08. IEEE, pp 160–164

- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. *HotCloud* 10(10–10):95
- Zhang K, Zhou X, Chen Y, Wang X, Ruan Y (2011) Sedic: privacy-aware data intensive computing on hybrid clouds. In: Proceedings of the 18th ACM conference on computer and communications security. ACM, pp 515–526
- Zhang X, Liu C, Nepal S, Dou W, Chen J (2012) Privacy-preserving layer over mapreduce on cloud. In: 2012 second international conference on cloud and green computing (CGC). IEEE, pp 304–310

Scalable SPARQL Query Processors

- ▶ Native Distributed RDF Systems

Scaling up OLTP on Multicores and Many Cores

- ▶ Hardware-Assisted Transaction Processing

Schema Mapping

Paolo Papotti
Campus SophiaTech – Data Science
Department, EURECOM, Biot, France

Synonyms

Mapping

Definitions

Several data management tasks require to precisely establish associations between data structured under different schemas. Schema mappings allow the definition of semantic connections between schemas with structural differences. A schema mapping M is a

specification of a relation between instances of a source schema S and instances of a target schema T . Given a source instance I for S and a target instance J for T , they satisfy the mapping if $(I, J) \models M$. Schema mappings are used both for virtual and materialized integration, and they have been studied in many aspects, including their specification, semantics, and user-assisted generation.

Overview

There are many applications that need to exchange, correlate, and integrate heterogeneous data sources. These information integration tasks have long been identified as important problems, and unifying theoretical frameworks have been advocated by database researchers (Bernstein and Melnik 2007). To solve these problems, a fundamental requirement is that of manipulating *mappings* among data sources. The application developer is typically given two schemas – one called the source schema S and the other called the target schema T – that can be based on radically different models and structures. *Schema mappings* are logical expressions that specify the semantic connection between the instances for the source and the target schemas. Given a source instance I and a target instance J that satisfy the schema mapping, we can say that $(I, J) \models M$.

A mapping is usually represented as a triple $\langle S, T, M \rangle$, and its semantics is defined with respect to the setting where only the source instance I (for S) is given. Before introducing the semantics, it is useful to distinguish the two main problems for which mappings play a key role: *data exchange* (Fagin et al. 2005a) and *data integration* (Lenzerini 2002). The two problems share the same goal: support users in retrieving source data by posing queries on the target, which is also known as global or mediated schema in data integration. The main difference is in the way that such goal is achieved. *Data exchange* formally studies the semantics of generating an instance of a target database given a source instance, a mapping, and constraints on the target schema. It has formalized the notion of a *data exchange*

problem and has established a number of results about its properties. In data exchange, the source data is transformed into a materialized instance that conforms to the target schema. Users define queries on the target schema, and these are then answered by using the target instance. *Data integration* is the problem of combining data residing at different sources and providing the user with a unified view of these data. The main difference with data exchange is that the query is part of the input and only the answer to such query is materialized. The source data is queried in situ, i.e., the target queries are rewritten to compute answers using the source data only. Despite the differences in the execution, in both approaches the goal is to reason about the possible instances J such that $(I, J) \models M$.

The most popular logical specification for schema mappings are source-to-target tuple-generating dependencies (s-t TGDs), with the following form:

$$\forall x (\Phi_S(x) \rightarrow \exists y \Psi_T(x, y)) \quad (1)$$

where $\Phi_S(x)$ and $\Psi_T(x, y)$ are conjunctions of atomic formulas over S and T , respectively. Consider, for example, a source schema with two relations *Patient* (*SSN*, *Name*, *Phone*, *HosDate*) and *Surgery* (*PatName*, *Insurance*, *Treatment*, *Date*) and the target schema with two relations *Customer* (*Id*, *Name*, *Phone*, *BirthDate*, *Insurance*) and *Claim* (*CustId*, *Treatment*, *Date*). A s-t TGD defined over these two schemas is

$$\begin{aligned} \forall p, i, t, d \text{ } & \text{Surgery}(p, i, t, d) \rightarrow \\ & \exists Y_1, Y_2, Y_3 (\text{Customer}(Y_1, p, Y_2, Y_3, i) \wedge \\ & \text{Claim}(Y_1, t, d)) \end{aligned}$$

Assume the TGD above is the only constraint in our mapping M . Consider now a given instance I for the source schema, such as $I = \text{Surgery}(\text{Mark}, \text{Axa}, \text{Eyes}, 2/21/17)$, and three target instances:

$$\begin{aligned} J_0 = & \text{Customer}(10, \text{Mark}, 978, 3/3/79, \text{Axa}), \\ & \text{Claim}(20, \text{Eyes}, 2/21/17) \end{aligned}$$

$$J_1 = \text{Customer}(11, \text{Mark}, 978, 3/3/79, \text{Axa}),$$

$$\text{Claim}(11, \text{Eyes}, 2/21/17)$$

$$J_2 = \text{Customer}(12, \text{Mark}, 978, 3/3/79, \text{Axa}),$$

$$\text{Claim}(12, \text{Eyes}, 2/21/17) \quad (2)$$

It is easy to see that the pair of instances (I, J_0) does *not* satisfy the s-t TGD specification, because of the different values for the customer id. In this case, we say that J_0 is not a *solution* for the given I and mapping M . On the other hand, both instances J_1 and J_2 satisfy the TGD together with I . Since $(I, J_1) \models M$ and $(I, J_2) \models M$, we say that each of the two target instances is a solution for I . One may observe that these two target instances have different customer ids, but this is not in contradiction with neither the source instance, which does not have this information, nor the mapping, as the s-t TGD only states that the id value should be the same in the *Customer* and *Claim* relations.

The example represents a mapping that specifies what source data must be in the solutions, but does not specify what must not be in any solution. For example, a fourth target instance J_3

$$\text{Customer}(12, \text{Mark}, 978, 3/3/79, \text{Axa}),$$

$$\text{Claim}(12, \text{Eyes}, 2/21/17),$$

$$\text{Claim}(12, \text{Liver}, 10/1/2010) \quad (3)$$

is also a solution for I and M above. The s-t TGDs above are in fact called *open* mappings and compose the most studied class of mappings in the literature (Bernstein and Melnik 2007). These mappings always allow a solution and can be specified with no requirements on the target or other sources (i.e., other open mappings cannot conflict). However, there are more constrained settings where exact mappings have been studied (Fuxman et al. 2006b).

Another important characterization of schema mappings is with respect to their structure (Lenzerini 2002). This comes from data integration systems, which assumed that the schema mapping is a function, e.g., a view. In one setting, known as global-as-view (GAV), mappings have

one relation symbol in $\Psi_T(x, y)$ and no repeated variables. In the opposite modeling, each relation of the source schema is defined in terms of the target schema. In this setting, known as local-as-view (LAV), mappings have one relation symbol in $\Phi_S(x)$ and no repeated variables. GAV and LAV have been largely studied and adopted because of their tractable data complexity when answering queries in data integration systems.

Key Research Findings

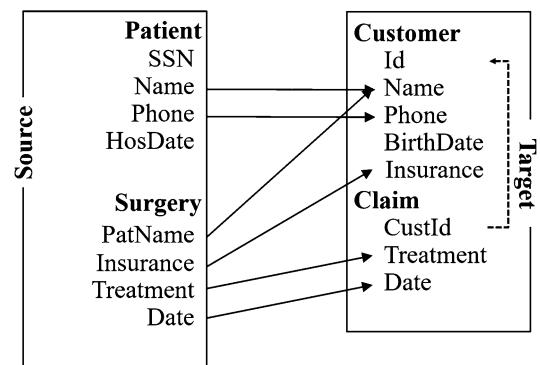
One important property of schema mappings is that they can be generated from graphical user interfaces (GUI). In fact, manually designing a schema mapping between schemas can be challenging. Even if the schemas represent similar data, they very often use different names and structures to describe the information. This is especially true when the schemas are independently designed. Manually crafting schema mappings is a difficult task, which requires technical experts that have deep understanding of both source and target schemas. When the size of the schemas increases, with dozens of relations and hundreds of attributes, it is also easy to make errors in the mappings, which can then lead to incorrect results in the ultimate application.

Since understanding the design of the schemas and manually writing schema mappings is time-consuming and prone to human errors, *mapping generation tools* have been created to make the process more abstract and user-friendly, thus easier to handle for a larger class of people. Most of these tools have been designed with data exchange in mind and directly compared to other solutions for the design of data transformation, such as ETL systems (Dessloch et al. 2008; Mecca et al. 2012). However, the same tools can be used to create schema mappings in data integration settings. Schema mapping generation tools include primarily Clio, which is the first tool able to generate logical mappings and corresponding scripts from high-level user input (Fuxman et al. 2006a; Haas et al. 2005;

Miller et al. 2000; Popa et al. 2002). Three main features characterize mapping generation tools.

High-level interface. The goal of simplifying the mapping specification was pursued by introducing a GUI that allows users to draw arrows, or *correspondences*, between schemas in order to define the desired transformation. Such correspondences can also be discovered and suggested to the users by schema matching algorithms (Bernstein et al. 2011). Correspondences represent associations between elements across different schemas in the form of attribute pairs and are discovered by profiling the attribute values and by mining the schema structures.

Consider the example shown in Fig. 1, with the relations described earlier and a foreign key from *Claim* to *Customer* in the target schema. Correspondences, represented as solid arrows in the figure, map atomic elements of the source schema to elements of the target schema, independently of the underlying data model or of the design choices. Due to the heterogeneity of the databases, the same concept could be represented differently across different schemas. At the same time, portions of the schemas with similar structure may actually model different entities. In the figure, the customer name in the target is encoded as the name of a patient in the source. However, multiple date attributes are present in both schemas, but they have different semantics across the relations, e.g., hospitalization date is



Schema Mapping, Fig. 1 Schema mapping scenario

different from birth year. While correspondences are easy to create and understand, they do not represent the full semantic relationship between source and target instances. In the example, the correspondences from the source relation *Surgery* to the target relations specify the constraint over the names and the treatments, but do not specify the relationship between the instances in relation *Customer* and *Claim*. This relationship is expressed explicitly by the foreign key and is part of the semantic specification, but it is lost in the attribute to attribute granularity of the correspondences. As we have seen in the schema mapping (2), a more complex formula is needed to capture precisely the semantic associations at the schema level. Correspondences are therefore taken as input by a schema mapping tool.

Automatic Mapping Generation. Based on value correspondences, mapping systems generate the logical dependencies that specify the mapping. In an operational interpretation, the resulting TGDs can be seen as a specification of how to *translate* data from the source to the target. It has been shown that the mapping generation is sound and complete with respect to the given correspondences, that is, all the relevant schema mappings are part of the output and none of them contradicts the correspondences (Fuxman et al. 2006a; Popa et al. 2002). Based on the correspondences drawn in Fig. 1, a mapping system would generate the following schema mapping with two dependencies:

SOURCE-TO-TARGET TGDS

$$\begin{aligned}
 m_1. & \forall s, n, p, h \text{ } Patient(s, n, p, h) \rightarrow \\
 & \exists Y_1, Y_2, Y_3 (\text{Customer}(Y_1, n, p, Y_2, Y_3)) \\
 m_2. & \forall p, i, t, d \text{ } Surgery(p, i, t, d) \rightarrow \\
 & \exists Y_1, Y_2, Y_3 (\text{Claim}(Y_1, p, Y_2, Y_3, i) \wedge \\
 & \quad \text{Account}(Y_1, t, d))
 \end{aligned} \tag{4}$$

Mapping Execution via Scripts. Given a source instance and a mapping, to produce a

target instance that is a solution, it suffices to execute the traditional *chase procedure* (Fagin et al. 2005a). The chase is a fixpoint algorithm which tests and enforces implication of data dependencies, such as TGDs, in a database. After the mappings had been generated, a schema mapping system translates the s-t TGDs under the form of an SQL script. When executed on any DBMS, the script implements the chase, i.e., it can be applied to a source instance I to create a new instance J that is a solution. Given a TGD, in order to chase it over I , we may see its $\Phi(x)$ as a first-order query Q_ϕ with free variables x over the source database. We execute $Q_\phi(I)$ using SQL in order to find all vectors of constants that satisfy the premise, and we then insert the appropriate tuple into the target instance to satisfy $\Psi(x, y)$. Skolem functions (Popa et al. 2002) are typically used to automatically generate new values for the existential variables y .

Given mappings written by users or generated from tools, a number of approaches have been proposed to *optimize* schema mappings in order to improve the efficiency of their execution and manipulation in real-world applications. In fact, schema mappings may present redundancy in their expressions, due, for example, to the presence of unnecessary atoms or unrelated variables, thus negatively affecting the data management process at hand (Fagin et al. 2008). The following efforts have aimed at optimizing such dependencies by identifying two kinds of equivalence aside from standard logical equivalence: the relaxed notions of data exchange (DE) equivalence and conjunctive query (CQ) equivalence. DE and CQ equivalences coincide with logical equivalence when the mapping scenario is made only of s-t TGDs, but differ on richer classes of equivalences, such as second-order TGDs, and scenarios with target constraints. Mapping rewriting has also been studied to compute a different class of data exchange solutions that reduces the redundancy in the target instances. Given a mapping scenario composed of s-t TGDs, these algorithms rewrite them to generate a runtime script that, on input instances, materializes compact solutions at scale (Mecca et al. 2009; Ten Cate et al. 2009). These works exploit the use of *negation*

in the premise of the s-t TGDs to rewrite them intercepting possible redundancy.

Schema mapping tools have proven to be effective in easing the burden of manually specifying TGDs and have been successfully transferred into commercial (Haas et al. 2005) and open-source systems (Marnette et al. 2011). Other commercial tools that generate some form of schema mappings from correspondences include Stylus Studio and Microsoft’s BizTalk Mapper and internal mapping modules in ETL systems. Given the large number of systems that have been proposed to create schema mappings, benchmarks for their systematic evaluation have been proposed (Alexe et al. 2008; Arocena et al. 2016).

Examples of Application

Schema mappings have been successfully adopted for data exchange and for data integration. They are popular because of their trade-off between expressiveness – as they can handle several real-world scenarios – and performance; formal results can be guaranteed with execution times that nicely scale over the size of the instances. However, many data management problems involve not only the design and execution of schema mappings but also their subsequent manipulation in a general framework (Bernstein and Melnik 2007). In this framework, other database applications, such as object-to-relational mappings and data warehousing, can be handled. This is achieved by introducing high-level algebraic operators that enable to build programs that directly manipulate schema mappings. One important application is *schema evolution*, where two of these operators play a key role. Consider a mapping M between schemas S and T , and assume that schema S evolves into a schema S' because of changes in the data organization or for performance reasons. The evolution can be expressed as a mapping M' from S to S' . Therefore, the relationship between the new schema S' and schema T can be obtained by first *inverting* the mapping M'

and then *composing* the result with the original mapping M . In this scenario inversion and composition are operators that take mappings as input and return new mappings. This high-level view of schema mappings exploits the intuition that they can be the building blocks for systems that save considerable effort to the final users.

Following this vision, the composition operator has been identified as one of the fundamental operators for the development of a framework for managing schema mappings, and its semantics has been studied (Fagin et al. 2005b). The main results show that first-order (FO) logic is not sufficient to express composition and that a second-order (SO) fragment of logic can be used as a mapping language for expressing this operator. Interestingly, SO TGDs can still be executed by means of scripts, and FO schema mappings can always be translated into this more general language, thus enabling composition. There are several other operators that have been defined for schema mappings, including several proposals on how to compute the inverse of a mapping (Arenas et al. 2009) and how to merge multiple mappings with the same target schema (Alexe et al. 2010).

Future Directions of Research

Emerging trends are encouraging the developing of more systems based on schema mappings. On one side, theoretical results are paving the way to the creation of innovative applications for which schema mappings show potential positive impact, such as data pipelines and graph data (Francis and Libkin 2017; Kolaitis et al. 2016). On the other side, new algorithms for the creation and optimization of schema mappings are widening the opportunities offered by such technology, with contributions such as mapping inference from data examples and probabilistic approaches (Ten Cate et al. 2017; Kimmig et al. 2017). These recent results show opportunities for enlarging the application of schema mappings to more data management tasks, including data fusion, data cleaning, and ETL scenarios.

Cross-References

► Data Integration

References

- Alexe B, Tan W, Velegrakis Y (2008) Comparing and evaluating mapping systems with STBenchmark. *PVLDB* 1(2):1468–1471
- Alexe B, Hernández MA, Popa L, Tan WC (2010) MapMerge: correlating independent schema mappings. *PVLDB* 3(1):81–92
- Arenas M, Pérez J, Reutter JL, Riveros C (2009) Composition and inversion of schema mappings. *SIGMOD Record* 38(3):17–28
- Arocena PC, Glavic B, Mecca G, Miller RJ, Papotti P, Santoro D (2016) Benchmarking data curation systems. *IEEE Data Eng Bull* 39(2):47–62
- Bernstein PA, Melnik S (2007) Model management 2.0: manipulating richer mappings. In: *SIGMOD*, pp 1–12
- Bernstein PA, Madhavan J, Rahm E (2011) Generic schema matching, ten years later. *PVLDB* 4(11):695–701
- Dessloch S, Hernandez MA, Wisnesky R, Radwan A, Zhou J (2008) Orchid: integrating schema mapping and ETL. In: *ICDE*, pp 1307–1316
- Fagin R, Kolaitis P, Miller R, Popa L (2005a) Data exchange: semantics and query answering. *TCS* 336(1):89–124
- Fagin R, Kolaitis P, Popa L, Tan W (2005b) Composing schema mappings: second-order dependencies to the rescue. *ACM TODS* 30(4):994–1055
- Fagin R, Kolaitis P, Nash A, Popa L (2008) Towards a theory of schema-mapping optimization. In: *ACM PODS*, pp 33–42
- Francis N, Libkin L (2017) Schema mappings for data graphs. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems. ACM, *PODS '17*, pp 389–401
- Fuxman A, Hernández MA, Howard CT, Miller RJ, Papotti P, Popa L (2006a) Nested mappings: schema mapping reloaded. In: *VLDB*, pp 67–78
- Fuxman A, Kolaitis PG, Miller RJ, Tan WC (2006b) Peer data exchange. *ACM Trans Database Syst* 31(4):1454–1498
- Haas LM, Hernández MA, Ho H, Popa L, Roth M (2005) Clio grows up: from research prototype to industrial tool. In: *SIGMOD*, pp 805–810
- Kimmig A, Memory A, Miller RJ, Getoor L (2017) A collective, probabilistic approach to schema mapping. In: 2017 IEEE 33rd international conference on data engineering (ICDE), pp 921–932
- Kolaitis PG, Pichler R, Sallinger E, Savenkov V (2016) Limits of schema mappings. In: 19th international conference on database theory (ICDT 2016), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Leibniz International proceedings in informatics (LIPIcs), vol 48, pp 19:1–19:17
- Lenzerini M (2002) Data integration: a theoretical perspective. In: *ACM PODS*, pp 233–246
- Marnette B, Mecca G, Papotti P, Raunich S, Santoro D (2011) ++SPICY: an opensource tool for second-generation schema mapping and data exchange. *PVLDB* 4(11):1438–1441
- Mecca G, Papotti P, Raunich S (2009) Core schema mappings. In: *SIGMOD*, pp 655–668
- Mecca G, Papotti P, Raunich S, Santoro D (2012) What is the IQ of your data transformation system? In: *CIKM*, pp 872–881
- Miller RJ, Haas LM, Hernandez MA (2000) Schema mapping as query discovery. In: *VLDB*, pp 77–99
- Popa L, Velegrakis Y, Miller RJ, Hernandez MA, Fagin R (2002) Translating web data. In: *VLDB*, pp 598–609
- Ten Cate B, Chiticariu L, Kolaitis P, Tan WC (2009) Laconic schema mappings: computing core universal solutions by means of SQL queries. *PVLDB* 2(1):1006–1017
- Ten Cate B, Kolaitis PG, Qian K, Tan W (2017) Approximation algorithms for schema-mapping discovery from data examples. *ACM Trans Database Syst* 42(2):12:1–12:41

SciDB

Philippe Cudre-Mauroux
eXascale Infolab, University of Fribourg,
Fribourg, Switzerland

Definitions

SciDB is a distributed database management system for managing and processing multidimensional arrays in scientific applications. It was first designed and developed by a group of academics led by Michael Stonebraker before being productized by Paradigm4.

Overview

The first XLDB workshop (1st Extremely Large Databases Workshop 2007) in 2007 brought together a group of scientists and industry members to discuss the capabilities of database management systems (DBMSs) at managing non-relational data at extreme scales. A number of

key shortcoming were identified, which were presented the following year at XLDB-2 (2nd Extremely Large Databases Workshop 2008). The lack of support for managing scientific data was in particular discussed. Consensus emerged that many Big Science projects presented unique challenges that could not be handled through generic DBMSs and would require a complete rewrite approach (Stonebraker et al. 2007).

As a result, a group of scientists led by Michael Stonebraker embarked on a new project to design and develop a new system for handling scientific arrays, as arrays represented a prominent data type for many science users (including astronomers, oceanographers, seismologists, or climate researchers). The resulting system, SciDB, was first presented at VLDB 2009 (Cudré-Mauroux et al. 2009). This first version supported a number of important features for science users, including:

1. native storage for nested, multidimensional arrays;
2. scientific operators built as user-defined functions (UDFs) to manipulate both the structure and the values of the arrays;
3. a shared-nothing design (Stonebraker 1986) allowing the system to scale out to many nodes and petabytes of data;
4. initial support for advanced features required by science users such as data versioning, provenance, and uncertainty.

The team continued the development of SciDB as an open source data management solution for Big Science projects. In 2011, Michael Stonebraker and Marilyn Matz co-founded Paradigm4 to further support the development of SciDB.

Architecture

SciDB is at its core a distributed array management system. The system takes as input large, multidimensional arrays potentially considering several values for each cell in the array. SciDB natively stores such arrays as a collection of *chunks*, each handling a portion of an array (e.g.,

a chunk or $10 \times 10 \times 5\text{ k}$ array cells). Chunks are typically of equal size (e.g., 64 MB) and store attribute values vertically (i.e., a chunk only stores the values of a given attribute).

Chunks partition the array spatially but can be overlapping to ease some operations like object detection, which would otherwise often involve stitching together multiple chunks. Chunks are written using various compression mechanisms on disk. The system adopts a *no-overwrite* storage strategy meaning that arrays cannot be updated in place (they can however be appended or updated by creating a new version of the array). A system catalog keeps track of the attributes and the spatial position of each chunk.

SciDB distributes the chunks among a set of worker nodes. Queries consist of a tree of operators over one or multiple arrays. Operators can modify both the structure of an array (e.g., its size or dimensions) as well as its contents (attribute values). The system comes with a number of standard operators such as *Filter*, which filters cell values based on a threshold, or *SJoin*, which combines attributes from different cells. The system is extensible and allows the definition of both user-defined types (UDTs) and user-defined functions (UDFs).

Many queries, such as *filter* or *object detection*, can be processed fully in parallel on each worker node. Specific operations might however require to exchange, share, or redistribute the chunks dynamically. SciDB defines a dedicated operator, called *ScatterGather*, to help support such cases. *Scatter/Gather* is a powerful operator allowing to dynamically redistribute the array over the nodes. Queries execution is orchestrated by a central node called the *coordinator*.

Academic Prototype

The first prototype of SciDB was developed as an open source project by a group of researchers from MIT, Brown University, SLAC, NIISI RAS, the University of Washington, Portland State University, and Microsoft. The group was led by Michael Stonebraker. The system was first demonstrated at VLDB in 2009 (Cudré-Mauroux et al. 2009).

This academic prototype adopted the architecture described above, with a non-overwrite, native array storage system and distributed query execution. It also featured initial support for array versioning, data provenance and uncertainty, as well as a simple query optimizer.

The main use case for this prototype was loosely modeled on an astronomy workload and was later extended to a full benchmark for scientific data management systems (Cudré-Mauroux et al. 2010). The use case considered the end-to-end ingestion and processing of raw data from a sensor system. It included three types of operations: (i) manipulation of raw imagery, including processing pixels to extract geo-spatial observations; (ii) manipulation of observations, including spatial aggregation and grouping into related sets; and (iii) manipulation of groups, including a number of relatively complex geometric operations in several dimensions.

The academic prototype was the basis of several research projects in array data management, including:

- A new storage manager to efficiently encode and access versioned arrays (Seering et al. 2012);
- A new storage manager for complex, parallel array processing implementing various partitioning and query execution strategies (Soroush et al. 2011);
- A hybrid analytic system for array-structured data integrating R and SciDB (Leyshock et al. 2013);
- A new structure to store and model multi-dimensional arrays supporting efficient inference processes (Ge and Zdonik 2010);
- Techniques to handle fine-grained lineage in scientific databases (Wu et al. 2013);
- Techniques to incrementally add nodes and to optimize data placement for n-dimensional array systems (Duggan and Stonebraker 2014).

Paradigm4 Development

In 2011, Michael Stonebraker and Marilyn Matz co-founded Paradigm4 to further support the development of SciDB (Paradigm4 2011).

The company hired Paul Brown to oversee the development of the system.

The system developed by Paradigm4 follows the architecture described above but adds a number of key features and libraries to SciDB (Stonebraker et al. 2011, 2013). It supports both a SQL-like, declarative language called *AQL* and a functional language (AFL). The system includes a full-fledged optimizer and an executor able to distribute queries to thousands of nodes. Support was also added for handling variable-size chunks, various encoding schemes (e.g., delta encoding, run-length encoding, or LZ encoding), uncertain data, and coarse-grained provenance.

A number of optimized operators come built-in with the system, including common linear algebra operators. The system also adds a number of dedicated operators for several vertical domain, e.g., for life sciences (to support genomic analysis, digital biomarker discovery, or biomedical images) and finance (for multifactor model generation or transaction cost analysis). Further use cases that have been explored include sensor analytics, insurance, and E-commerce.

Cross-References

- [Architectures](#)

S

References

- 1st Extremely Large Databases Workshop (2007) <http://www-conf.slac.stanford.edu/xldb2007/>
- 2nd Extremely Large Databases Workshop (2008) <http://www-conf.slac.stanford.edu/xldb2008/>
- Cudré-Mauroux P, Kimura H, Lim K, Rogers J, Simakov R, Soroush E, Velikhov P, Wang DL, Balazinska M, Becla J, DeWitt DJ, Heath B, Maier D, Madden S, Patel JM, Stonebraker M, Zdonik SB (2009) A demonstration of SciDB: a science-oriented DBMS. PVLDB 2(2):1534–1537. <http://www.vldb.org/pvldb/2/vldb09-76.pdf>
- Cudré-Mauroux P, Kimura H, Lim KT, Rogers J, Madden S, Stonebraker M, Zdonik SB (2010) Ss-db: A standard science DBMS benchmark. http://www-conf.slac.stanford.edu/xldb10/docs/ssdb_benchmark.pdf

- Duggan J, Stonebraker M (2014) Incremental elasticity for array databases. In: International conference on management of data, SIGMOD 2014, Snowbird, pp 409–420. <https://doi.org/10.1145/2588555.2588569>
- Ge T, Zdonik SB (2010) A*-tree: a structure for storage and modeling of uncertain multidimensional arrays. PVLDB 3(1):964–974. <http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R86.pdf>
- Leyshock P, Maier D, Tufte K (2013) Agrios: a hybrid approach to big array analytics. In: Proceedings of the 2013 IEEE international conference on big data, Santa Clara, pp 85–93. <https://doi.org/10.1109/BigData.2013.6691558>
- Paradigm4 (2011) <https://www.paradigm4.com/>
- Seering A, Cudré-Mauroux P, Madden S, Stonebraker M (2012) Efficient versioning for scientific array databases. In: IEEE 28th international conference on data engineering (ICDE 2012), Washington, DC (Arlington), pp 1013–1024. <https://doi.org/10.1109/ICDE.2012.102>
- Soroush E, Balazinska M, Wang DL (2011) Arraystore: a storage manager for complex parallel array processing. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2011, Athens, pp 253–264. <https://doi.org/10.1145/1989323.1989351>
- Stonebraker M (1986) The case for shared nothing. IEEE Database Eng Bull 9(1):4–9. <http://sites.computer.org/debull/86MAR-CD.pdf>
- Stonebraker M, Madden S, Abadi DJ, Harizopoulos S, Hachem N, Helland P (2007) The end of an architectural era (it's time for a complete rewrite). In: Proceedings of the 33rd international conference on very large data bases, University of Vienna, pp 1150–1160. <http://www.vldb.org/conf/2007/papers/industrial/p1150-stonebraker.pdf>
- Stonebraker M, Brown P, Poliakov A, Raman S (2011) The architecture of SciDB. In: Scientific and Statistical Database Management – proceedings of the 23rd international conference, SSDBM 2011, Portland, pp 1–16. https://doi.org/10.1007/978-3-642-22351-8_1
- Stonebraker M, Brown P, Zhang D, Becla J (2013) SciDB: a database management system for applications with complex analytics. Comput Sci Eng 15(3):54–62. <https://doi.org/10.1109/MCSE.2013.19>
- Wu E, Madden S, Stonebraker M (2013) Subzero: a fine-grained lineage system for scientific databases. In: 29th IEEE international conference on data engineering, ICDE 2013, Brisbane, pp 865–876. <https://doi.org/10.1109/ICDE.2013.6544881>

SDM Theory

► Spatial Data Mining

Search and Query Accelerators

Johns Paul¹, Bingsheng He², and Chiew Tong Lau¹

¹Nanyang Technological University, Singapore, Singapore

²National University of Singapore, Singapore, Singapore

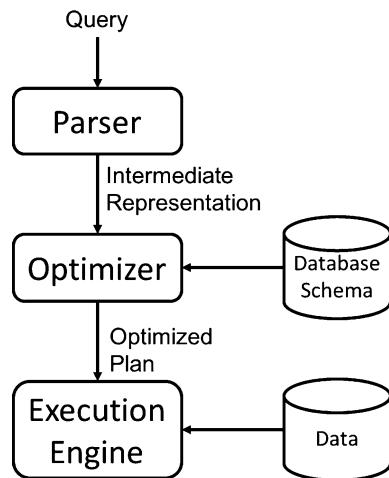
Search and Query Processing

Search and query processing involves translation of high-level queries or user requests into low-level operations which can be executed on physical hardware to retrieve relevant results from a database. The process generally involves conversion of the high-level query into an intermediate representation, optimization of the intermediate representation to generate an optimal evaluation plan and finally the execution of the optimized plan on physical hardware to retrieve relevant results (Markl 2009). Figure 1 shows the major steps involved in query processing for relational databases.

The performance of search and query processing is essential for our daily life and work productivity. Besides relational databases, everyday applications of search and query processing range from simple Google search to complex deep learning systems. Other common examples include online shopping, supporting internal operations of organizations, and even simple file searches on personal computers.

Accelerators for Query Processing

Most current query processing systems made use of CPUs to evaluate queries submitted by users. However, the explosive growth in the amount of available data and the huge demand for high-throughput and low-latency query processing have resulted in the adoption of specialized hardware devices to accelerate query processing. These accelerators act as an add-on to CPUs and



Search and Query Accelerators, Fig. 1 Query processing steps

are used only for running specialized tasks like query processing. Some of the most common accelerators used for query processing include GPUs, FPGAs, and x86 based accelerators like Xeon Phi.

GPU

Graphics processing units (GPUs) were initially designed to accelerate the rendering of images for video editing and gaming. However more recently, GPUs have evolved as a powerful accelerator for processing huge amounts of data in parallel. There are already a number of query processing systems available for GPUs including GPUQP (Fang et al. 2007), Ocelot (Heimel et al. 2013), GPL (Paul et al. 2016), MapD (MapD 2016), and Voodoo (Pirk et al. 2016).

Hardware Design

Similar to CPUs, GPUs also contain processing cores, registers, multiple levels of cache, and a global memory unit (the equivalent of the main memory for CPUs). However, GPUs differ from CPUs in terms of the design of each one of these components. Instead of a small number of complex cores available on the CPUs, GPUs use a large number of relatively simpler cores that work in a SIMD fashion to process large amounts of data. For example, the latest Pascal GPUs from NVIDIA contain more than 3500 individual cores

which can work in parallel. This is more than 100 \times the number of cores in a modern CPU. GPUs are able to fit such a large number of cores in a single silicon die due to the relative simplicity of each individual core. These cores lack complex components such as branch prediction unit and even have much smaller cache than CPUs. However, they do have much larger number of registers and a higher bandwidth global memory unit, which is usually smaller (in size) than the main memory available on CPUs. The High Bandwidth Memory (HBM) units available in modern GPUs can provide close to 10 \times the bandwidth of the main memory accessible to CPUs. This high bandwidth and large number of cores help the modern Pascal GPUs from NVIDIA to achieve up to 9 TFLOPs of compute power using a single GPU. Figure 2 shows an overview of the internal architecture of the latest Pascal generation of GPUs from NVIDIA (2016).

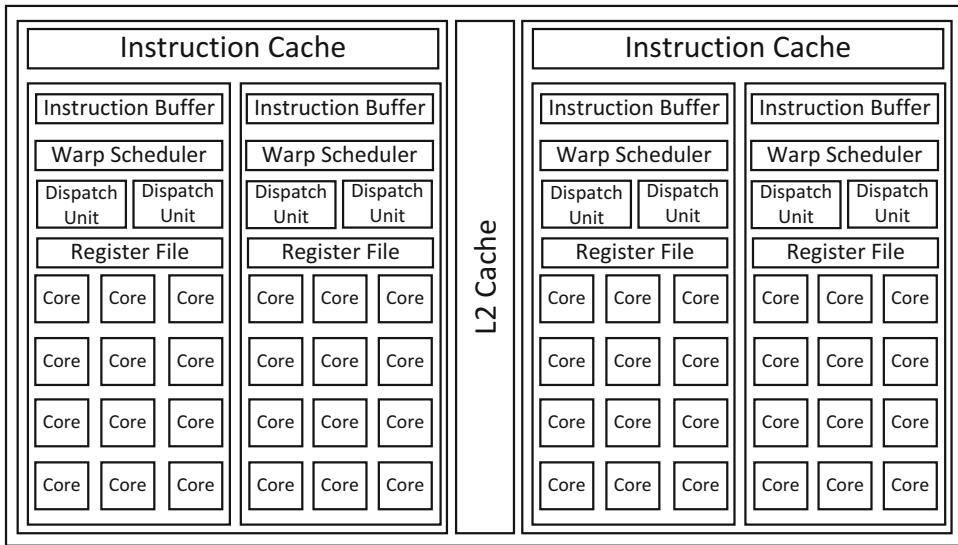
Most modern GPUs are available as a separate hardware unit which can be connected to the CPU over the PCIe bus. Any data that needs to be processed by the GPU needs to be first copied to the GPU global memory over the PCIe bus. However, the 16 GB/s memory bandwidth of the current generation of PCIe bus (3.0) has proved to be a bottleneck to processing large amounts of data on GPUs. Hence, vendors like NVIDIA have started looking into better interconnects like NVLink which is capable of providing up to 5x the bandwidth of the PCIe bus.

S

Advantages and Limitations

The main advantage of GPUs for query processing tasks comes in the form of the availability of large number of parallel cores which work in a SIMD fashion. This is mostly because query processing operations largely involve applying the same operation to a large number of data items in parallel. Further, the availability of larger number of registers and the much higher global memory bandwidth of GPUs ensure fast data access to the large number of parallel cores, thus making it possible to achieve much better throughput than CPUs.

However, the need to use specialized languages like CUDA or OpenCL to program the



Search and Query Accelerators, Fig. 2 GPU internal architecture

GPUs have proved to be a major limiting factor in ensuring widespread adoption of GPUs. This is because existing query processing systems need to be rewritten in these languages to take advantage of GPUs. Further, the low bandwidth of the PCIe bus also dampens the overall performance of GPU-based systems due to the high overhead of data movement between the CPU main memory and GPU global memory. This coupled with the relatively small size of GPU memory limits the performance gains achieved by addition of GPUs to large query processing systems.

FPGA

Field-programmable gate arrays (FPGAs) are specialized semiconductor devices that consist of a collection of logic blocks which can be connected together using programmable interconnects based on the operation that needs to be executed on the FPGA. Due to its unique hardware design, FPGAs can achieve significantly better energy efficiency than other hardware accelerators and even CPUs when performing most tasks. This has led to services like Microsoft Bing search adopting FPGAs in their servers (Allison Linn 2016).

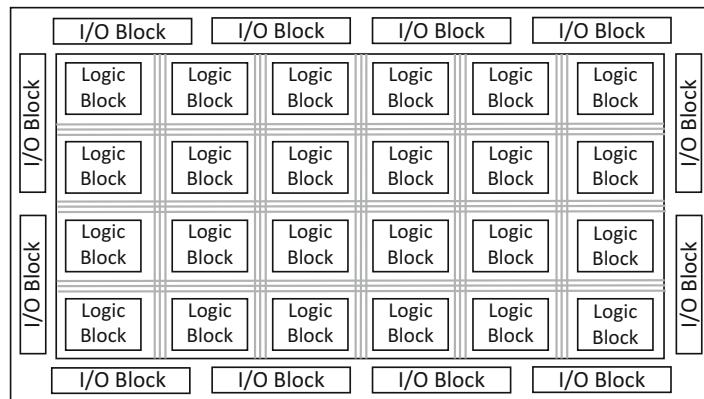
Hardware Design

An FPGA hardware consists of four main hardware units: logic blocks, registers, block RAM, and digital signal processors (DSPs). These hardware devices can be connected together based on the computation that needs to be executed on the FPGA. This means that FPGAs execute operations based on the interconnection in the hardware instead of decoding individual instructions as in the case of other hardware devices like CPUs or GPUs. Hence, FPGAs have the ability to process data more efficiently than other hardware devices making them significantly more energy efficient. FPGAs can be connected to CPUs over PCIe bus or Ethernet. Figure 3 shows an overview of architectural design of FPGAs.

In spite of being highly energy efficient, FPGA hardware usually falls behind other accelerators like GPUs when it comes to operating frequency and global memory bandwidth. Even though modern FPGAs have started adopting High Bandwidth Memory instead of traditional DDR RAM, these systems are still unable to achieve the same level of bandwidth as GPUs. FPGAs also lack additional on-chip cache which significantly impacts its performance when running programs written in high-level languages like OpenCL. Recent studies (Wang et al. 2016;

Search and Query Accelerators, Fig. 3

FPGA internal architecture



Woods et al. 2014) have demonstrated some promising results on accelerating query and search performance on FPGAs.

Advantages and Limitations

The major advantage of using FPGAs for query processing is its lower power consumption, which is very important in large data warehouses which spend a significant amount of money on power and cooling. Further, the use of hardware interconnections and the reliance on pipeline parallelism allow FPGAs to process data with much lower latency than other hardware devices like CPUs and GPUs. This is especially beneficial when querying data streams.

The major limitation of using FPGAs is the difficulty involved in programming them. Most FPGAs can be programmed only using low-level hardware description languages (HDLs) like Verilog or VHDL, making it extremely tedious to develop and maintain large-scale query processing systems on FPGAs. FPGA vendors have recently tried to address this issue by releasing OpenCL-based FPGAs which can be programmed using high-level languages like OpenCL. However, converting an operator written in OpenCL into an FPGA hardware implementation takes hours, and debugging these applications is extremely tedious. Further, the limited amount of resources available on the FPGA makes it difficult to fit a large number of operators on a single FPGA. All this makes it difficult to adopt FPGAs for systems that need to constantly adapt to user queries or changes in data streams.

Xeon Phi

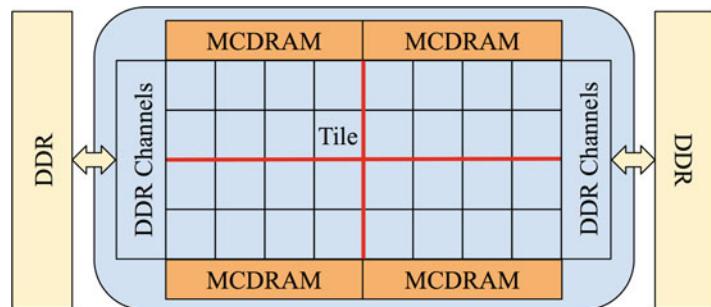
The Xeon Phi accelerator was first introduced in 2012 by Intel as an alternative to accelerators like GPUs. The attraction of Xeon Phi is the use of traditional x86 cores which allow the use of standard programming languages and APIs such as OpenMP. This means that most existing query processing implementations designed for CPUs can be ported over to the Xeon Phi accelerator without the need for any modification. Due to the use of traditional x86 architecture, the latest generation of Xeon Phi can even run an operating system and work without CPUs.

Hardware Design

The Xeon Phi follows a many-core design that uses the x86 architecture. The accelerator tries to offer a device which contains much larger number of cores than modern CPUs while being architecturally similar to existing CPUs. The recent release of Xeon Phi, known as Knights Landing (Avinash Sodani 2016), contains over 70 cores which make use of SIMD intrinsics to process large amounts of data in parallel. These cores are arranged in a 2D mesh and have access to L1 and L2 cache, but it lacks an L3 cache. Further, due to power and area limitations, each individual core has much lower operating frequency and a much simpler pipeline design when compared to modern CPUs. The Xeon Phi accelerator also has access to high-speed memory stacks integrated into the same silicon die. Figure 4 shows the

Search and Query Accelerators, Fig. 4

Xeon Phi internal architecture



architectural design of the Knights Landing processor.

Further, in addition to operating as a stand-alone device (without the need for a CPU), the Xeon Phi can also be used in conjunction with traditional CPUs by connecting it to the CPU over the PCIe bus. Recent studies have demonstrated significant performance improvement by tuning and optimization on those platforms (Jha et al. 2015; Cheng et al. 2017).

Advantages and Limitations

The major advantage of the Xeon Phi accelerator is the much larger number of x86 cores available for processing data in parallel. This design enables the execution of existing query processing systems on the Xeon Phi accelerator without any modification and at the same time makes it possible to achieve much higher levels of parallelism than traditional CPUs.

However, the simpler design and lower operating frequency of each individual core limit the performance of the accelerator when executing single-threaded workload. This is especially problematic due to the relatively low performance of individual cores. The lack of an L3 cache can also have a significant negative impact in such cases. Further, connecting the Xeon Phi to the CPU over PCIe bus leads to the PCIe interconnect becoming a bottleneck for data transfer.

Future Hardware

A recent trend in query processing hardware seems to be the development of coupled architectures which consist both CPUs and specialized accelerators in the same physical die or

on the same physical board. In such cases the accelerators are connected to CPUs using much faster interconnects like QPI or NVLink. Such a design allows the specialized hardware to access data residing in the system main memory without being bottlenecked by the lower-bandwidth PCIe bus. Some designs even allow the specialized hardware to access the L3 cache of CPUs, allowing for more fine-grained and faster movement of data between CPUs and hardware accelerators. Another recent trend in this area seems to be the emergence of database processing units (DPUs), which are specialized devices with hardware support for execution of relational operators (Wu et al. 2014).

References

- Allison Linn (2016) The moonshot that succeeded: how bing and azure are using an AI supercomputer in the cloud. https://blogs.microsoft.com/ai/2016/10/17/the_moonshot_that_succeeded
- Avinash Sodani (2016) Knights landing (KNL): 2nd generation Intel®Xeon Phi processor. <https://www.alcf.anl.gov/files/HC27.25.710-Knights-Landing-Sodani-Intel.pdf>
- Cheng X, He B, Du X, Lau CT (2017) A study of main-memory hash joins on many-core processor: a case with intel knights landing architecture. In: Proceedings of the 2017 ACM on conference on information and knowledge management, CIKM '17. ACM, New York, pp 657–666. <http://doi.acm.org/10.1145/3132847.3132916>
- Fang R, He B, Lu M, Yang K, Govindaraju NK, Luo Q, Sander PV (2007) GPUQP: query co-processing using graphics processors. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data, SIGMOD '07. ACM, New York, pp 1061–1063. <http://doi.acm.org/10.1145/1247480.1247606>
- Heimel M, Saecker M, Pirk H, Manegold S, Markl V (2013) Hardware-oblivious parallelism for in-memory

- column-stores. Proc VLDB Endow 6(9):709–720. <http://dx.doi.org/10.14778/2536360.2536370>
- Jha S, He B, Lu M, Cheng X, Huynh HP (2015) Improving main memory hash joins on intel xeon phi processors: an experimental approach. Proc VLDB Endow 8(6):642–653. <http://dx.doi.org/10.14778/2735703.2735704>
- MapD (2016) The world's fastest platform for data exploration. <http://go.mapd.com/rs/116-GLR-105/images/MapD%20Technical%20Whitepaper%20Summer%202016.pdf>
- Markl V (2009) Query processing (in relational databases). Springer US, Boston, pp 2288–2293. https://doi.org/10.1007/978-0-387-39940-9_296
- NVIDIA (2016) NVIDIA Tesla P100. <https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf>
- Paul J, He J, He B (2016) GPL: a GPU-based pipelined query processing engine. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. ACM, New York, pp 1935–1950. <http://doi.acm.org/10.1145/2882903.2915224>
- Pirk H, Moll O, Zaharia M, Madden S (2016) Voodoo – a vector algebra for portable database performance on modern hardware. Proc VLDB Endow 9(14):1707–1718. <http://dx.doi.org/10.14778/3007328.3007336>
- Wang Z, Paul J, Cheah HY, He B, Zhang W (2016) Relational query processing on opencl-based fpgas. In: 2016 26th international conference on field programmable logic and applications (FPL), pp 1–10. <http://dx.doi.org/10.1109/FPL.2016.7577329>
- Woods L, István Z, Alonso G (2014) Ibex: an intelligent storage engine with support for advanced sql offloading. Proc VLDB Endow 7(11):963–974. <http://dx.doi.org/10.14778/2732967.2732972>
- Wu L, Lottarini A, Paine TK, Kim MA, Ross KA (2014) Q100: the architecture and design of a database processing unit. SIGARCH Comput Archit News 42(1):255–268. <http://doi.acm.org/10.1145/2654822.2541961>

Secure Big Data Computing in Cloud: An Overview

Sambit Kumar Mishra, Sampa Sahoo, and Bibhudatta Sahoo

National Institute of Technology Rourkela,
Rourkela, India

Definitions

Cloud computing: It is a system model that provides services on demand from a shared pool of

resources (CPU, main memory, secondary storage, bandwidth, etc.) without violating service level agreement (SLA) and maintaining a certain level of quality of service (QoS). *SLA:* The agreement done between the cloud service provider and cloud user before accessing the services.

Big data: It is an emerging area applied to store, manage, and analyze the data whose volume is large.

Virtualization: It is a technology that facilitates multiple virtual machines (VMs) over a single physical machine (host) with the help of a hypervisor or virtual machine monitor (VMM).

Virtual machine: A VM is a software program that emulates a real (physical) computing environment where an operating system is installed and can run different applications.

Introduction

The big data (explosion of data) problem aroused over the last 5–8 years as a result of the wide area network access, proliferation of next-generation applications, and advent of social media. Diverse sectors like finance, retail, smart sensor networks (IoT), physical and life science, etc., generate large data sets that need significant storage and computational implications. The bursty nature of data set spurred the use of cloud computing, where users can rent infrastructure on a “pay-as-you-go” basis. Virtualization technology, the basis of cloud computing, customized hardware and computational power of physical machines (PM), thus creating an illusion of several machines (virtual machines (VM)). Virtualization ensures the maximum of resource utilization and capital investment (Mishra et al. 2018b). A VM is a software application that emulates a physical computing environment, and multiple VMs can run on a single physical machine. Thus, virtualization reduces large capital infrastructure and maintenance cost. Cloud computing provides a scalable and cost-efficient solution to the big data challenge. Traditional computing (grid) technology had put a significant effort on resilience and robustness instead of solving the actual problem.

Modern big data technologies use scalable and cost-efficient methods to overcome such limitations.

The big data technology, though extremely powerful, was built for the technically savvy and needs applications to be parallelized in nature (ODriscoll et al. 2013). Big data with cloud technology makes several applications (e.g., biological problem) into reality to provide solutions. The implication of cloud computing to big data is due to the scalable and fault tolerance features of cloud computing toward big data computing. Addressing big data is a challenging and time-demanding task that requires a sizeable computational infrastructure to ensure reliable data processing and analysis. Since users are transmitting their data through the Internet, it gives rise to privacy issues, concerning profiling, stealing, and loss of control (Puthal et al. 2015). Although cloud computing transformed modern ICT technology, there exist security threats that can be elaborated by the volume, velocity, and variety of big data (Hashem et al. 2015).

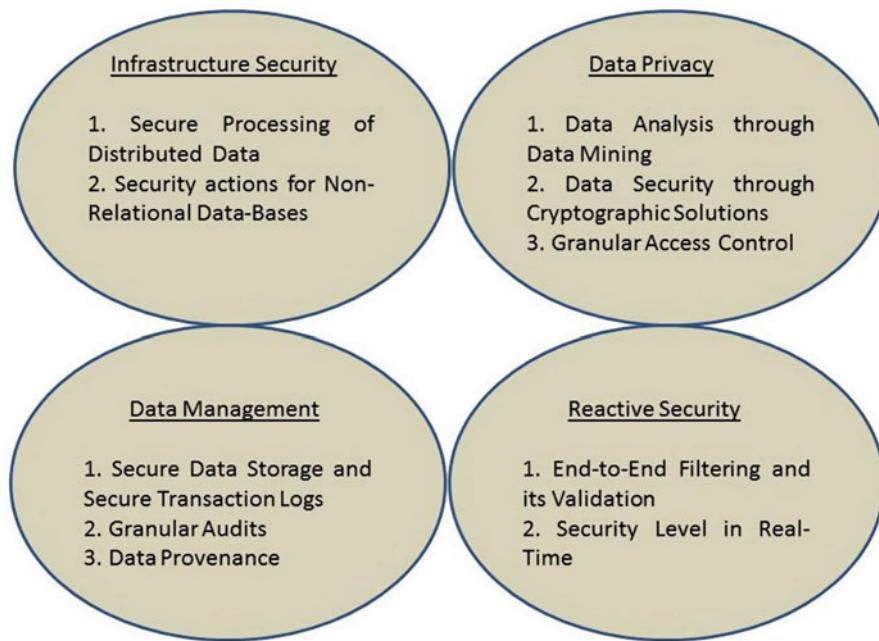
Some drawbacks associated with utilization of cloud computing are (1) data transfer rate: data from some applications may take long transmission time, even a week (e.g., genomic data) over the Internet. (2) Security and protection: Even though cloud computing is capable of handling big data sets, there is a loophole in terms of security and protection of data. Many applications such as cloud-based health-care system must provide enough security and protection of sensitive data from intruders. (3) Data tenancy: cloud infrastructures provide very little capability on data, application, and service back to an on-premise IT environment.

The big data processing capabilities of cloud computing generates a new area of computing system research. The requirement of secured big data computing in cloud is more essential due to the rise of the Internet of things (IoT). Gartner estimates that 26 billion of IoT devices (connected in mobiles, cars, TVs, security systems, etc.) will be installed by 2020 (Rivera and van der Meulen 2014). This will be responsible for a significant security challenge; however, it presents huge benefits for end users.

The big data not only deal with terabytes of data but also need to handle the issue of managing data under a traditional framework. The high-speed connectivity era permit movement of large sets of data that may contain sensitive information like credit/debit card numbers, addresses, and other details, raising data security concerns. Security issues in the cloud are a major concern for businesses and cloud providers today. The remaining of the paper is arranged as follows. The next section outlines an overview of secured challenges of big data computing in cloud; after that the next section reviews various solutions for security challenges in cloud followed by the conclusion and future direction.

Secured Challenges of Big Data Computing in Cloud

The complex infrastructure of cloud computing system increases various challenges for the IT industries and researchers. The volume of data collection, storage, and processing in the cloud system increases day by day, which brings new challenges in terms of the information security. Due to the stretching of security mechanisms out of the perimeter of the requirements, general firewalls cannot be used in the big data infrastructure (Kune et al. 2016). The challenges for secure big data computing may be organized into different big data aspects such as infrastructure security, data privacy, data management, and reactive security (Moura and Serrão 2016). Each of these mentioned security aspects face different security challenges as presented in Fig. 1. The security functions are compelled to operate over the heterogeneous composition using big data. Different technologies introduced abstraction that can enable the big data secure services on top of the heterogeneous infrastructure and separate the control from the system infrastructure. The production rate of data has been rising exponentially due to various sources like sensors. Thus, data privacy plays an important role due to the risk of erroneous data in every field. For example, the patient data; the content of patient medical record possess high-risk on the health-care system. The



Secure Big Data Computing in Cloud: An Overview, Fig. 1 Four different aspects of secure challenges of big data computing in cloud

big data computing has a set of risk areas that includes the ownership and classification of data, the process of creation, and collection of data. It would be more challenging in terms of research point of view to develop techniques to delegate encrypted data so that third parties can analyze it.

Some sensitive data require more security for the sake of the client; there should be an agreement (i.e., specified in SLA) upon the location of data, as its data may be considered illegal for others and lead to prosecution. Data encryption is required to solve security and privacy issues. Therefore, researchers have focused on the generation of new systems that must ensure the quick access of data where there is no effect of encryption on processing. To improve the system performance, various mechanisms (task allocation, VM consolidation, etc.) has to be improvised (Mishra et al. 2018a). To optimize the performance parameters like energy consumption, makespan, throughput, SLA violation, etc., the profit for the CSP as well as for the cloud user reduced (Mishra et al. 2018b). Due to the complexity of the system with the aim of optimizing various performance

parameters, the security level also has to be improved, which leads to a huge cost.

Solutions to the Security Challenge

For an efficient big data management and processing, the cloud system needs a high-speed transport mechanism that addresses two significant bottlenecks: (1) the degradation in the speed of WAN transfer that occurs over distance through traditional transfer protocols and (2) the last foot bottleneck caused by the HTTP interfaces in the cloud data center to the underlying object based cloud storage.

The new era of computing technology allows companies to store and analyze the vast amount of data (e.g., company, business, customer). So, it becomes a challenging task to make the data secure. To make the big data secure, various techniques are used for encryption, honeypot detection, logging, etc. The challenge of detecting and preventing advanced threats and malicious intruders must be solved using big data style analysis. With the rise in the use of big data,

companies must have to deal with both security and privacy issues. The big data issues are most acutely felt in certain industries, such as telecoms, web marketing and advertising, retail and financial services, and certain government activities. So, resource allocation and memory management algorithms used in the cloud for big data also have to be secure. Numerous data mining techniques can be used in the malware detection in cloud (Inukollu et al. 2014).

Kune et al. have elaborated the differences between the traditional data warehousing and big data and also discussed various solutions for the security issues in big data computing (Kune et al. 2016). Several areas of big data computing (i.e., scientific explorations, health care, governance) have been explained in Kune et al. (2016). The collected data from different sensors are extracted by analyzing those data for societal benefits.

Attackers also keep inventing new ideas of attacking to a secure system. Other issues like ransomware profoundly affect a company's reputation and resources, denial of service attacks, phishing attacks, and cloud abuse. Globally, 40% of businesses experienced a ransomware incident during the past year. Both clients and cloud providers have their own share of risks involved when agreeing on cloud solutions. Insecure interfaces and weak APIs can give away valuable information to hackers, which can be misused. Fraud detection patterns, encryptions, and smart solutions are immensely valuable to combat attackers. At the same time, it is the responsibility of the user to own data and keep it safe while looking for intelligent answers that can assure a steady ROI (return on investment) as well (Riaz 2017).

Conclusion and Future Directions

This chapter presents some of the most significant security and privacy issues that affect big data computing in cloud environment and also discusses the information security, methodologies, and tools to provide security and privacy to the system. It also presents some relevant solutions for these security issues.

This chapter does not point to directions and technologies that contribute to solve some of the relevant security issues.

References

- Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU (2015) The rise of big data on cloud computing: review and open research issues. *Inf Syst* 47:98–115
- Inukollu VN, Arsi S, Ravuri SR (2014) Security issues associated with big data in cloud computing. *Int J Netw Secur Appl* 6(3):45
- Kune R, Konugurthi PK, Agarwal A, Chellarige RR, Buyya R (2016) The anatomy of big data computing. *Softw: Pract Experience* 46(1):79–105
- Mishra SK, Puthal D, Rodrigues JJ, Sahoo B, Dutkiewicz E (2018a) Sustainable service allocation using metaheuristic technique in fog server for industrial applications. *IEEE Trans Ind Inf*
- Mishra SK, Puthal D, Sahoo B, Jena SK, Obaidat MS (2018b) An adaptive task allocation technique for green cloud computing. *J Supercomput* 74:370–385
- Moura J, Serrão C (2016) Security and privacy issues of big data. *arXiv preprint arXiv:160106206*
- ODriscoll A, Daugelaite J, Sleator RD (2013) Big data, Hadoop and cloud computing in genomics. *J Biomed Inform* 46(5):774–781
- Puthal D, Sahoo B, Mishra S, Swain S (2015) Cloud computing features, issues, and challenges: a big picture. In: Computational intelligence and networks (CINE), 2015 international conference on. IEEE, pp 116–123
- Riaz K (2017) <http://bigdata-madesimple.com/big-data-and-cloud-computing-challenges-and-opportunities/>
- Rivera J, van der Meulen R (2014) Gartner says the internet of things will transform the data center. Retrieved 5 Aug 2014

Security and Privacy Aspects of Semantic Data

Sabrina Kirrane

Vienna University of Economics and Business,
Vienna, Austria

Synonyms

[Security and privacy for the resource description framework](#)

Definitions

Access control is a mechanism used to restrict access to data or systems, based on rules that grant *subjects* (e.g., individuals, groups, roles) *access rights* to *resources* (e.g., data or systems) (Sandhu and Samarati 1994). Enforcement is usually broken into two stages: *authentication* and *authorization*. Authentication involves the verification the data subjects identity or attributes, whereas authorization is a mechanism used to determine if the requester (i.e., the subject) has the access rights necessary to carry out the request.

Encryption is an effective means of ensuring the confidentiality and integrity of information stored locally or transferred over a network (Menezes et al. 1996). Encryption involves the translation of data into an unintelligible form through the use of a secret key. Decryption is the process of restoring data to its original form through the use of a key (which may or may not be the same as the key used to encrypt the data). Encrypted data is referred to as cipher or cipher text, whereas unencrypted data is commonly known as plain text.

Trust mechanisms are used to verify the validity of a claim (e.g., the identity/attributes of an individual or the correctness of data). The most widely used trust mechanisms include *policies* and *reputation* (Artz and Gil 2007). Policies are used to govern the exchange of credentials that are often certified by trusted third parties. Reputation mechanisms take the form of provenance information and metrics that are calculated from previous actions and behaviors. Where no such data is available, trust may be established via referral from other trusted parties.

Anonymization involves the removal of personally identifiable information from datasets. One of the most well-known anonymization techniques, k-anonymity, involves the use of suppression (i.e., masking sensitive data) and generalization (i.e., choosing broader classification terms for sensitive data), in order to group individuals into equivalence classes, whereby each individual in a class is indistinguishable from *k-1* other individuals (Samarati and Sweeney 1998).

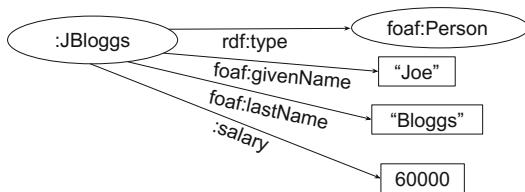
Overview

The Resource Description Framework (Manola and Miller 2004) is designed to facilitate data integration and reuse by representing distributed data in a machine-readable format. RDF *vocabularies* (otherwise known as *ontologies*) are collections of RDF triples that can be used to describe both schema and instance data. Each triple, which is composed of a *subject-predicate-object* expression, asserts a binary relationship between two pieces of information. *Internationalized Resource Identifiers (IRIs)* and *literals* are used to represent information, which can be either physical or abstract in nature. The *RDFSchema* (RDFS) ontology (Brickley and Guha 2014) is composed of a set of classes and properties commonly used to describe RDF data. RDFS does not describe the structure of an RDF graph but rather provides a framework that can be used to denote classes, properties, and relations. Vocabularies are often placed in a common *namespaces*, referenced via *prefixes*. In the examples that follow, the default prefix `:` is used to denote an example enterprise ontology `<http://example.org/ex/>`. In addition, well-known `rdf` and `foaf` prefixes are used for the RDF built-in vocabulary and FOAF social network ontology, respectively. Example 1 demonstrates how RDF can be used to represent information pertaining to Joe Bloggs.

Example 1 (RDF triples) The following triples state that the entity `:JBloggs` is a person whose first name is `Joe`, last name is `Bloggs`, and salary is `60,000`:

```
:JBloggs rdf:type foaf:Person.  
:JBloggs foaf:givenName "Joe".  
:JBloggs foaf:lastName "Bloggs".  
:JBloggs :salary 60000.
```

An *RDF graph* is a finite set of RDF triples, with subjects and objects represented as nodes and predicates represented as edges. Figure 1 demonstrates how the triples in Example 1 converge to form a graph, with IRIs represented as ovals and literals represented as rectangles. A collection of RDF graphs, which can include a



Security and Privacy Aspects of Semantic Data, Fig. 1
Triples represented as an RDF graph

default graph and one or more named graphs, is known as an RDF dataset.

In a recent survey by Fernandez Garcia et al. (2016), the authors analyzed topics appearing in papers that were published in Semantic Web conference proceedings and journals from 2006 to 2015 inclusive. The results of the conducted text analysis confirmed that traditional Semantic Web topics, such as knowledge representation, data management, system engineering, searching, browsing and exploration, and data integration, dominated the field up to 2015.

According to Fernandez Garcia et al. (2016), although topics relating to security and privacy have shown a minor increase over the years, the topics remain under-represented in comparison to traditional topics.

Much of the early research on security and privacy in the context of the Semantic Web focused on using RDF to represent existing access control models and standards and demonstrating how the technology could be used to develop general policy languages. Later the focus moved to the development of access control strategies for RDF and the Semantic Web. Other popular topics over the years include demonstrating how existing encryption mechanisms can be used to protect RDF data and establishing trust mechanisms for the Semantic Web. More recently, the landscape has broadened to include the encryption and anonymization of RDF data.

Key Research Findings

The goal of this section is to introduce the reader to key research findings in relation to Semantic Web security and privacy, and as such it focuses

on the predominant topics within the community, namely, access control, encryption, trust, and anonymization.

Access Control

Access Control (AC) for the RDF data model has predominately focused on using patterns to specify authorizations, enabling inference based on semantic relations between policy entities, and demonstrating how RDF can be used to form general policy languages.

Reddivari et al. (2005) demonstrate how access control rules can be used to manage access to an RDF store. Two predicates permit and prohibit are used to grant and deny access rights based on common database actions (e.g., INSERT, DELETE, SELECT) to one or more triples using triple patterns (cf. Example 2). A triple pattern is composed of an RDF triple with optionally a variable (denoted by a ?) in the subject, predicate, and/or object position.

Example 2 (AC rules with triple patterns) The following rule states that a subject Alice can create instances of any class (denotes as ?y) if there is an assertion that subject Alice created that class.

```

permit (INSERT(Alice,
(?x,rdf:type,?y)))
:- createdNode(Alice,?y)
  
```

Jain and Farkas (2006) build on the approach proposed by Reddivari et al. (2005), by demonstrating how RDFS entailment rules can be used to derive authorizations for inferred triples. While Kirrane et al. (2013) demonstrate how authorizations based on quad patterns (where the fourth element denotes the named graph) can be used to enforce Discretionary Access Control (DAC), where users can pass their access rights on to other users. Like Jain and Farkas (2006), the authors derive access rights for derived data using RDFS entailment rules.

When it comes to access control enforcement, typical enforcement strategies involve filtering unauthorized data based on access control policies and executing queries against the

filtered dataset or using query rewriting techniques to inject access control filters into queries.

Dietzold and Auer (2006) and Gabillon and Letouzey (2010) demonstrate how graph patterns (i.e., sets of triple patterns) constrained by a WHERE clause can be used to create a new dataset that only contains data the subject is permitted to access. The authorized dataset is created using SPARQL the standard query language for RDF (Seaborne and Prud'hommeaux 2008). Essentially, authorizations contain filters that refer to sparql CONSTRUCT queries that are used to generate the authorized dataset. In Gabillon and Letouzey (2010) a rule such as Permit(Alice, SELECT, foafview.txt) can be used to permit subject Alice, access right SELECT on resource foafview.txt. The resource foafview.txt simply contains a CONSTRUCT query such as that presented in Example 3). When a requester submits a query, a new dataset is created based on the matched authorizations. The query is executed against the new dataset, which only contains data that the requester is permitted to access.

Example 3 (Construct view) The following query creates a dataset that contains all data relating to people with Bloggs as a last name.

```
CONSTRUCT {?x ?p ?y}
WHERE {
?x ?p ?y .
?x foaf:lastName "Bloggs"}
```

An alternative enforcement strategy proposed by Abel et al. (2007) uses query rewriting to create bindings for variables that are subsequently added to the query WHERE clause. In the case of negative authorizations, the bindings are added to a MINUS clause, which is appended to the query. A simple SPARQL SELECT query is presented in Example 4, and sample rewritten queries containing positive and negative filters are presented in Examples 5 and 6, respectively. When a requester submits a query, the enforcement framework rewrites the query according to the matching authorizations, and the rewritten query is

subsequently executed against the new dataset, ensuring that the requester is only returned data that they are permitted to access.

Example 4 (SELECT query) The following query returns all data.

```
SELECT *
WHERE { ?s ?p ?o }
```

Example 5 (Positive filter) The following query, which contains a positive filter, only returns the information for :JBloggs.

```
SELECT *
WHERE { ?s ?p ?o .
FILTER ( ?s = :JBloggs ) }
```

Example 6 (Negative filter) The following query, which contains a negative filter, returns everything except the :salary information.

```
SELECT *
WHERE { ?s ?p ?o .
MINUS { ?s ?p ?o .
FILTER ( ?p = :salary ) }}
```

In addition to the access control mechanisms described above, there have been a number of standardization initiatives that could be used to limit access to RDF data. *Web Identity and Discovery (WebID)* (Sporny et al. 2011) is a mechanism that can be used to uniquely identify and authenticate a person, company, organization, or other entity, by means of a Uniform Resource Identifier (URI), while *Web Access Control (WAC)* W3C (n.d.) is an RDF vocabulary and access control framework that can be used for policy specification and enforcement. Both Villata et al. (2011) and Sacco and Passant (2011) extend WAC to cater for access control over the RDF data model. Using the extended vocabularies, it is possible to associate access control with individual RDF resources (subjects, predicates, and objects) and also collections of RDF resources (named graphs). In addition, the authors extend the vocabulary to cater for a broader set of access privileges.

An alternative policy language, called the Open Digital Rights Language (ODRL) (Iannella

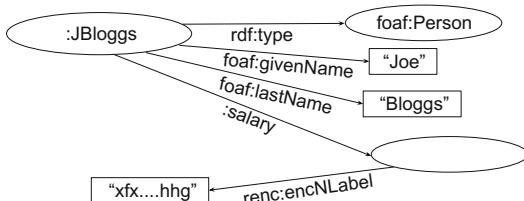
and Villata 2018), is a general rights language that can be used to define rights for limiting access to digital resources. When it comes to ODRL and RDF, primary research efforts to date focus on demonstrating how ODRL can be used to express a variety of access policies (Steyskal and Polleres 2014; Steyskal and Kirrane 2015) and using ODRL vocabularies to specify RDF licenses (Cabrio et al. 2014).

A comprehensive survey of existing access control strategies for RDF is presented in Kirrane et al. (2017).

Encryption

Encryption techniques for RDF have received very little attention to date, with work primarily focusing on the partial encryption of RDF data, the querying of encrypted data, and the signing of RDF graphs.

Giereth (2005) demonstrates how public-key encryption can be used to partially encrypt RDF fragments (i.e., subjects, objects, or predicates). The ciphertext and the corresponding metadata (algorithm, key, hash, etc.) are represented using a literal that they refer to as an encryption container. When only the object is encrypted, the object part of the triple is replaced with a blanknode (i.e., an anonymous resource), and a new statement is created with the blanknode as the subject, the encryption container as the object, and a new `renc:encNLabel` as the predicate (cf. Fig. 2). The treatment of encrypted subjects is analogous. The encryption of predicates is a little more difficult, as reification (a technique used to make statements about resources) is needed



Security and Privacy Aspects of Semantic Data, Fig. 2
Partially encrypted RDF graph

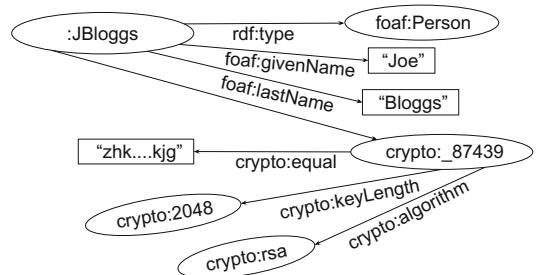
to associate the new blanknode with the relevant subject, object, and encryption container.

Rather than simply storing the encrypted data and metadata in a literal, Gerbracht (2008) discusses how the metadata that can be represented using multiple triples using their crypto ontology. The encrypted element or subgraph is replaced with a new unique identifier, and new statements are added for the encrypted data and the corresponding metadata (cf. Fig. 3).

Kasten et al. (2013) in turn focus on querying encrypted data. In the proposed framework, each triple is encrypted eight times according to the eight different triple pattern binding possibilities. The proposed approach allows for graph pattern queries to be executed over the ciphertext, at the cost of storing multiple ciphers for each statement. An alternative approach by Fernández et al. (2017) demonstrates how functional encryption can be used to generate query keys based on triple patterns, whereby each key can decrypt all triples that match the corresponding triple pattern. Other work by Kasten et al. (2014) investigates enabling of the signing of graph data at different levels of granularity.

Trust

In 2007, Artz and Gil (2007) conducted a survey of existing trust mechanisms in computer science in general and the Semantic Web in particular. The authors highlight that traditional approaches focused primarily on authentication via assertions by third parties; however, in later years, the topic evolved to include historical interaction data, the transfer of trust from trusted entities,



Security and Privacy Aspects of Semantic Data, Fig. 3
Partially encrypted RDF graph and metadata

and decentralized trust mechanisms (e.g., voting mechanisms or other consensus decision-making mechanisms).

Existing work on trust and semantic data focuses primarily on demonstrating how existing trust metrics can be applied to Semantic Web Data, the development of policy languages to support trust and negotiation, and the identification of trust architectures and frameworks.

Ding et al. (2003, 2005) discuss how various trust mechanisms can be combined in order to determine the reliability of information published on the Semantic Web. The proposed trust mechanism combines historical data, information obtained directly from trusted semantic agents, and information based on referrals from trusted agents.

The PeerTrust policy language and framework (Gavriloaie et al. 2004) demonstrate how together semantic annotations and access control rules can be used to support automated trust negotiation and access control. An alternative policy language called Protune is described in Bonatti and Olmedilla (2005, 2007). Although Protune is in fact a general policy language, the authors focus primarily on trust negotiation and policy explanations.

Bizer and Oldakowski (2004) propose a trust architecture that combines reputation, content, and context-based trust mechanisms. The *information integration layer* aggregates data from several sources and adds the relevant provenance metadata. The *repository layer* is used to store the information and associated metadata in named graphs. The *query and trust evaluation layer* uses trust policies to make trust decisions. Here the authors rely on a query language TriQL.P that is used to return the query results together with a justification tree that can be used to understand how the query results fulfill the trust requirement. Finally the *application and explanation layer* receives requests and provides the trust decision together with the relevant explanations.

More recently, Laufer and Schwabe (2017) propose a framework that can be used to describe the trust process. Inputs to be considered include the data and associated metadata and contextual

information relating to the action that needs to be taken, together with trust policies specified by the agent. Like Ding et al. (2003), the trust process relies on historical data, along with direct and indirect sources of information.

Anonymization

The anonymization of RDF data has recently emerged as a popular research topic, with work to date focusing on the application of k-anonymity (Samarati and Sweeney 1998) or differential privacy (Dwork 2006) to RDF data.

Radulovic et al. (2015) propose a framework called k-RDFanonymity, which includes an anonymization model, generalization and suppression operations, and distortion metrics that are specifically tailored for the anonymization of RDF data. The authors highlight the fact that RDF differs from tabular data as identifiers, quasi-identifiers, and sensitive attributes can appear in the subject, predicate, and object positions. Additionally the anonymization needs to be able to handle data represented as literals and IRIs. In the proposed model, generalization involves the replacement of resources (i.e., literals or IRIs) with more general resources based on domain hierarchies, while suppression involves either the removal or replacement of resources.

Heitmann et al. (2017) build on this work to ensure protection against neighborhood attacks. The proposed approach, which is known as k-RDF-Neighborhood anonymity, ensures that one-hop neighbors of an anonymized resource are indistinguishable from k-1 one-hop neighbors of other resources.

Other work in relation to RDF anonymization includes adopting graph or statistical database approaches. Lin (2016) takes inspiration from existing graph isomorphism-based anonymization techniques, discussing their suitability for RDF data from both a security and a computational complexity perspective. Whereas, Silva et al. (2017) explore the application of existing differential privacy mechanism to RDF data and propose a framework that can be used to compute differential privacy parameters.

Examples of Application

Semantic Web technologies have a solid foundation in open standards as evidenced by the various World Wide Web Consortium (W3C) recommendations; however, the layers of the Semantic Web technology stack (Berners-Lee 2000) that relate to security and privacy (i.e., unifying logic, proof, trust, and cryptography) are still very immature. Although the application of the key research findings described in the previous section is still very exploratory, several of the articles are guided by real-world use cases and practical applications.

For instance, the Protune policy language (Bonatti and Olmedilla 2005, 2007), which was developed by the Research Network of Excellence on Reasoning on the Web, known as REWERSE, was tasked with building the foundations for the advanced of Web systems and applications by developing inter-operable reasoning languages.

Fernández et al. (2017) are motivated by a real word use case that involves the combination of open and closed data in a data market scenario. In order to demonstrate the suitability of the proposed encryption mechanism, the authors conduct a performance evaluation over two real-world datasets: Jamendo a large dataset containing licensed music and the AEMET metereological dataset.

Although the initial evaluation of the trust framework proposed by Ding et al. (2003) was conducted using simulated data, the authors later discussed how trust mechanisms could be used in the context of homeland security, in order to identify suspicious individuals, relationships, activities, or events (Ding et al. 2005). Similarly, Laufer and Schwabe (2017) describe how the proposed trust framework can be used to evaluate the trustworthiness of claims in relation to political agents in Brazil coming from a variety of public sources (e.g., news stories, tweets, social media postings).

Existing work on anonymization appears to be less applied than the other topics with authors simply motivating their work by referring to privacy concerns in domains, such as healthcare and energy (cf. Radulovic et al. 2015).

Future Directions for Research

From a community perspective, it is well known that privacy is a multidisciplinary research area, which brings with it the need for closer collaboration between computer scientists, humanities, and social scientists and legal scholars. Although initiatives such as the *Society, Privacy and the Semantic Web – Policy and Technology* (PrivOn) workshop, which was collocated with the International Semantic Web Conference (ISWC) from 2013 to 2017, provides a forum for multidisciplinary research, stronger collaboration between different research communities is still needed.

From a technical perspective, there is a need for more applied work and a focus on attacker models across all privacy and security topics. Additionally there are many open research questions concerning the topics presented in this paper, several of which are outlined below.

When it comes to information security, there is still no standard access control strategy for the Semantic Web. Considering the array of access control specification and enforcement mechanisms proposed to date, a necessary first step is to develop a framework that can be used to evaluate existing offerings in terms of correctness, completeness, and robustness.

As for encrypted RDF, it is still not possible to execute complex queries and computations over encryption RDF data. One interesting avenue for future work is the application of homomorphic encryption to RDF; however, it brings with it performance and scalability issues that still need to be tackled. Another open research topic is the simplification of key management for multiple datasets, federated querying over encrypted data, and providing support for the revocation of existing keys.

In terms of trust, a recent article by Beek et al. (2016) highlights several issues with respect to the quality of existing data and datasources, claiming that the Semantic Web is neither traversable nor machine-processable and consequently arguing that the Semantic Web needs centralization. A counterargument, which is more in keeping with the goals of the Semantic

Web, would be to argue for the application of trust mechanisms into the fabric of the Semantic Web, which could be brought about by the realization of the upper layers and vertical layers of the Semantic Web technology stack.

Anonymization is a relatively new topic within the Semantic Web community with works primarily focusing on k-anonymity. However, it is well known that k-anonymity is prone to homogeneity and background knowledge attacks. Common extension mechanisms include l-diversity (Li et al. 2007), which ensures that sensitive attributes within an equivalence class are suitably different, and t-closeness, which ensures that the distribution of each equivalence class is representative of the distribution of the entire dataset (Machanavajjhala et al. 2006).

Other promising privacy and security research directions that remain underdeveloped and as such have not been presented in this article include usage control, which is defined as an extension of access control that enables data publishers to dictate not only who can access their data but also what they are permitted to do with this data (Bonatti et al. 2017). Related topics include transparency, which involves being open with respect to data processing and sharing, and accountability, which involves making data consumers responsible for their actions. Here, interesting avenues for future work include the adoption and extension of non-repudiation and fair exchange protocols.

Cross-References

- ▶ [Big Data for Cybersecurity](#)
- ▶ [Privacy-Aware Identity Management](#)
- ▶ [Privacy-Preserving Data Analytics](#)
- ▶ [Privacy-Preserving Record Linkage](#)

References

Abel F, De Coi J, Henze N, Koesling A, Krause D, Olmedilla D (2007) Enabling advanced and context-dependent access control in RDF stores. In: The semantic web. Lecture notes in computer science, vol 4825.

- Springer, Berlin/Heidelberg, pp 1–14. https://doi.org/10.1007/978-3-540-76298-0_1
- Artz D, Gil Y (2007) A survey of trust in computer science and the semantic web. *Web Semant Sci Serv Agents World Wide Web* 5(2):58–71
- Beek W, Rietveld L, Schlobach S, van Harmelen F (2016) Lod laundromat: why the semantic web needs centralization (even if we don't like it). *IEEE Internet Comput* 20(2):78–81
- Berners-Lee T (2000) Semantic web – xml2000. <https://www.w3.org/2000/Talks/1206-xml2k-tbl/slides/10-0.html>. Accessed 13 Jan 2018
- Bizer C, Oldakowski R (2004) Using context-and content-based trust policies on the semantic web. In: Proceedings of the 13th international World Wide Web conference on alternate track papers & posters. ACM, pp 228–229
- Bonatti P, Olmedilla D (2005) Driving and monitoring provisional trust negotiation with metapolices. In: Sixth IEEE international workshop on policies for distributed systems and networks, pp 14–23
- Bonatti PA, Olmedilla D (2007) Rule-based policy representation and reasoning for the semantic web. In: Proceedings of the third international summer school conference on reasoning web, RW'07. Springer, pp 240–268. <http://dl.acm.org/citation.cfm?id=2391482.2391488>
- Bonatti P, Kirrane S, Polleres A, Wenning R (2017) Transparent personal data processing: the road ahead. In: International conference on computer safety, reliability, and security. Springer, London/New York, pp 337–349
- Brickley D, Guha R (2014) RDF schema 1.1. W3C recommendation, W3C. Available at <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/Overview.html>
- Cabrio E, Aprosio AP, Villata S (2014) These are your rights a natural language processing approach to automated RDF licenses generation. In: The semantic web: trends and challenges. Springer, Cham, pp 255–269
- Dietzold S, Auer S (2006) Access control on RDF triple stores from a semantic wiki perspective. In: Proceedings of the ESWC'06 workshop on scripting for the semantic web
- Ding L, Zhou L, Finin TW (2003) Trust based knowledge outsourcing for semantic web agents. In: Proceedings IEEE/WIC international conference on web intelligence, pp 379–387
- Ding L, Kolari P, Finin T, Joshi A, Peng Y, Yesha Y (2005) On homeland security and the semantic web: a provenance and trust aware inference framework. In: AAAI spring symposium: AI technologies for homeland security, pp 157–160
- Dwork C (2006) Differential privacy. In: Proceedings of the 33rd international conference on automata, languages and programming – volume part II, ICALP'06. Springer, Berlin/Heidelberg, pp 1–12. https://doi.org/10.1007/11787006_1
- Fernández JD, Kirrane S, Polleres A, Steyskal S (2017) Self-enforcing access control for encrypted RDF. In: European semantic web conference. Springer, pp 607–622

- Fernandez Garcia JD, Kiesling E, Kirrane S, Neuschmid J, Mizerski N, Polleres A, Sabou M, Thurner T, Wetz P (2016) Propelling the potential of enterprise linked data in Austria. Roadmap and report. https://www.linked-data.at/wp-content/uploads/2016/12/propel_book_web.pdf
- Gabilon A, Letouzey L (2010) A view based access control model for sparql. In: 2010 4th international conference on network and system security (NSS), pp 105–112
- Gavriloaie R, Nejdl W, Olmedilla D, Seamons KE, Winslett M (2004) No registration needed: how to use declarative policies and negotiation to access sensitive resources on the semantic web. In: ESWWS. Springer, pp 342–356
- Gerbracht S (2008) Possibilities to encrypt an RDF-graph. In: Proceeding of information and communication technologies: from theory to applications, pp 1–6
- Giereth M (2005) On partial encryption of RDF-graphs. In: Proceeding of international semantic web conference, vol 3729, pp 308–322
- Heitmann B, Hermsen F, Decker S (2017) k-RDF-neighbourhood anonymity: combining structural and attribute-based anonymisation for linked data. In: Proceedings of the 5th workshop on society, privacy and the semantic web – policy and technology (PrivOn2017) (PrivOn). <http://ceur-ws.org/Vol-1951/#paper-03>
- Iannella R, Villata S (2018) ODRL information model 2.2. W3C proposed recommendation, W3C. Available at <https://www.w3.org/TR/odrl-model/>
- Jain A, Farkas C (2006) Secure resource description framework: an access control model. In: Proceedings of the eleventh ACM symposium on access control models and technologies, SACMAT '06. ACM, pp 121–129. <http://doi.acm.org/10.1145/1133058.1133076>
- Kasten A, Scherp A, Armknecht F, Krause M (2013) Towards search on encrypted graph data. In: Proceeding of the international conference on society, privacy and the semantic web-policy and technology, pp 46–57
- Kasten A, Scherp A, Schaub P (2014) A framework for iterative signing of graph data on the web. Springer International Publishing, Cham, pp 146–160. https://doi.org/10.1007/978-3-319-07443-6_11
- Kirrane S (2015) Linked data with access control. PhD thesis, INSIGHT Centre for Data Analytics, National University of Ireland, Galway. <https://aran.library.nuigalway.ie/handle/10379/4903>
- Kirrane S, Abdelrahman A, Mileo A, Decker S (2013) Secure manipulation of linked data. In: The semantic web – ISWC 2013. Lecture notes in computer science, vol 8218. Springer, Berlin/Heidelberg, pp 248–263. https://doi.org/10.1007/978-3-642-41335-3_16
- Kirrane S, Mileo A, Decker S (2017) Access control and the resource description framework: a survey. Semant Web 8(2):311–352. <http://www.semantic-web-journal.net/system/files/swj1280.pdf>
- Laufer C, Schwabe D (2017) On modeling political systems to support the trust process. In: Proceedings of the 5th workshop on society, privacy and the semantic web – policy and technology (PrivOn2017) (PrivOn). <http://ceur-ws.org/Vol-1951/#paper-07>
- Li N, Li T, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: IEEE 23rd international conference on data engineering, ICDE 2007. IEEE, pp 106–115
- Lin Z (2016) From isomorphism-based security for graphs to semantics-preserving security for the resource description framework (RDF). Master's thesis, University of Waterloo
- Machanavajjhala A, Gehrke J, Kifer D, Venkitasubramaniam M (2006) l-diversity: privacy beyond k-anonymity. In: Proceedings of the 22nd international conference on data engineering, ICDE'06. IEEE, pp 24–24
- Manola F, Miller E (2004) RDF primer. W3C recommendation, W3C. Available at <http://www.w3.org/TR/rdf-primer/>
- Menezes AJ, Van Oorschot PC, Vanstone SA (1996) Handbook of applied cryptography. CRC press, Boca Raton
- Radulovic F, García Castro R, Gómez-Pérez A (2015) Towards the anonymisation of RDF data. <https://doi.org/10.18293/SEKE2015-167>
- Reddivari P, Finin T, Joshi A (2005) Policy-based access control for an RDF store. In: Proceedings of the policy management for the web workshop, pp 78–83
- Sacco O, Passant A (2011) A privacy preference ontology (PPO) for linked data. In: Linked data on the web, CEUR-WS. <http://ceur-ws.org/Vol-813/lidow2011-paper01.pdf>
- Samarati P, Sweeney L (1998) Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International
- Sandhu RS, Samarati P (1994) Access control: principle and practice. IEEE Commun Mag 32(9):40–48
- Seaborne A, Prud'hommeaux E (2008) SPARQL query language for RDF. W3C recommendation, W3C. Available at <http://www.w3.org/TR/rdf-sparql-query/>
- Silva RRC, Leal BC, Brito FT, Vidal VMP, Machado JC (2017) A differentially private approach for querying RDF data of social networks. In: Proceedings of the 21st international database engineering & applications symposium, IDEAS 2017. ACM, New York, pp 74–81. <https://doi.acm.org/10.1145/3105831.3105838>
- Sporny M, Inkster T, Story H, Harbulot B, Bachmann-Gmr R (2011) WebID 1.0 – web identification and discovery. W3C working draft, W3C. Available at <http://www.w3.org/2005/Incubator/webid/spec/>
- Steykal S, Kirrane S (2015) If you can't enforce it, contract it: enforceability in policy-driven (linked) data markets. In: SEMANTiCS (posters & demos), pp 63–66
- Steykal S, Polleres A (2014) Defining expressive access policies for linked data using the odrl ontology 2.0. In: Proceedings of the 10th international conference on semantic systems. ACM, pp 20–23
- Villata S, Delaforge N, Gandon F, Gyrard A (2011) An access control model for linked data. In: On the move

to meaningful internet systems: OTM 2011 workshops, pp 454–463
W3C (n.d.) Webaccesscontrol. Available at <https://www.w3.org/wiki/WebAccessControl>. Accessed 13 Jan 2018

Security and Privacy for the Resource Description Framework

► Security and Privacy Aspects of Semantic Data

Security and Privacy in Big Data Environment

Shekha Chenthara, Hua Wang, and Khandakar Ahmed

Institute for Sustainable Industries and Liveable Cities, VU Research, Victoria University, Melbourne, Australia

Synonyms

Attribute-based access control (ABAC); Authentication, authorization, and accounting (AAA); Availability; Confidentiality; Distributed denial of service (DDoS); Electronic health data (EHD); Electronic health records (EHR); Integrity; Privacy, Authentication, Integrity, Nonrepudiation (PAIN); Ramp secret sharing scheme (RSSS); Role-based access control (RBAC)

Format Definition

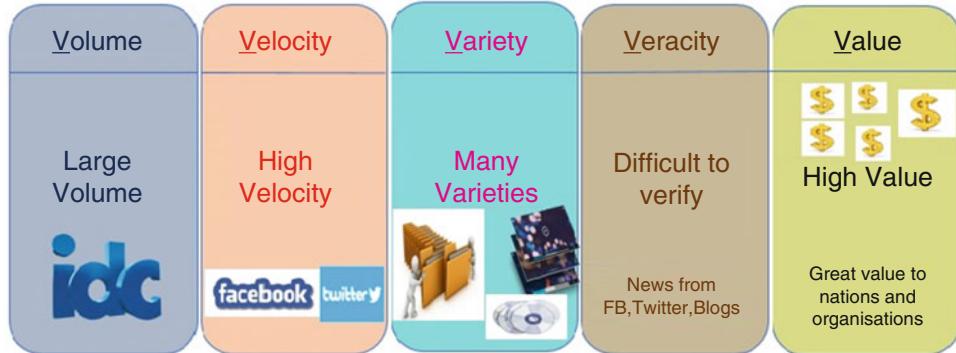
We come across data in every possible form, whether through social media sites, sensor networks, digital images or videos, cell phone GPS signals, purchase transaction records, weblogs, medical records, archives, military surveillance, e-commerce, complex scientific research, and numerous fields. This amount could reach to some quintillion bytes of data! This data is what we call ... BIG DATA! (Venkatram and Geetha 2017). “Big data is nothing but an assortment of huge

and complex data that it becomes very tedious to capture, store, process, retrieve and analyze with the help of on-hand database management tools or traditional data processing techniques.” We live in the Age of Big Data where everything that surrounds us is connected to a data source and everything is captured digitally (Matturdi et al. 2014). In the past few years, the total amount of data created by human has been exploded (McCune 1998). From 2005 to 2020, the amount of data is predicted to increase 300 times, from 130 exabytes to 40,000 exabytes (Gantz and Reinsel 2012). These data are generated by scientific research, finance and business informatics, government, Internet search, social networks, document, photography, audio, video, logs, click streams, mobile phones, sensor networks, and so on, and Big Data is the result of this dramatic increase of data.

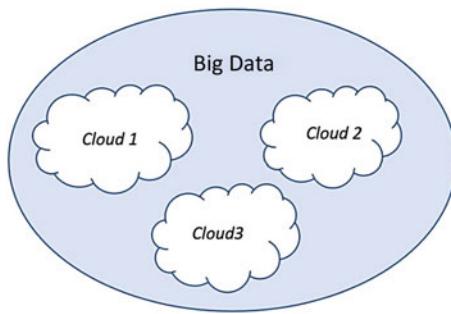
Overview of Big Data

Big Data can be represented by 5Vs – volume, velocity, variety, veracity, and value (Kadhiwala 2017). It is defined by five characteristics of Big Data, including five levels of meaning. Initially, volume refers to huge amount of data. Secondly, velocity refers to fast processing speed. Thirdly, variety refers to different varieties of data categories. Big Data is taken from multiple data sources, and data types contain structured, semi-structured, and unstructured data, such as network log, video, pictures, geographic location information, and so on. Fourthly, veracity refers to the authenticity of data. Finally, value refers to the worth of Big Data (Fang et al. 2017) (Fig. 1).

As the amount of data is increasing day by day, *cloud* has become a perfect solution to store data by providing virtually unlimited storage that can be accessed over the Internet. By outsourcing large volume of data to cloud storage, such as Google Drive, Dropbox, and Amazon Simple Storage Service, users can simplify their data management and reduce data maintenance costs through the pay-as-you-use model (Shu et al. 2018). Cloud services provide the necessary infrastructure by reducing the cost of storing, processing, and updating information with improved



Security and Privacy in Big Data Environment, Fig. 1 5 V features of Big Data



Security and Privacy in Big Data Environment, Fig. 2
An overview of Big Data

efficiency and quality. However, privacy of Big Data is a major hurdle while outsourcing private data in third-party *cloud* as there is a possibility of leaking/sharing sensitive information contained in Big Data with unauthorized entities. As most of the data is sensitive and strictly confidential, security of stored data is a major concern in Big Data environment. Since Big Data often contains sensitive information that needs to be protected from unauthorized access, therefore release of several techniques has to be devised immediately to protect it (Fig. 2).

The content of the chapter is enumerated as follows: Section “Format Definition” describes an overall view of big data. Section “Historical Background” describes challenges and issues of big data security and privacy. Section “Need for Data Security” reveals the need for big data security via some real examples. Section “Key Applications” describes one of the applications

related to big data security and privacy and some solutions to enforce big data security and privacy. Section “Future Research Directions” briefly introduces future research areas, and finally, Section “Conclusion” concludes this chapter.

Historical Background

The term “Big Data” is used to indicate the exponential growth and availability, the variety of data, and the speed at which the data is produced and transferred. The rise of Big Data contributes enormous opportunities for individuals, organizations, and society (Huang et al. 2017). An important notion is privacy for Big Data, since it contains sensitive information about individuals. Several privacy models, such as k-anonymity (Sweeney 2002), l-diversity (Machanavajjhala et al. 2007), t-closeness (Li et al. 2007), and differential privacy (Dwork 2011), can be used to anonymize data. It also raises some privacy and ethical issues. Big Data brings some challenges such as heterogeneity, data life cycle management, data processing, scalability, data visualization, security and privacy, etc.

Challenges of Big Data Security and Privacy

Undoubtedly the most challenging and concerned problem in Big Data is security and privacy. Governmental agencies, healthcare industry, biomedical researchers, and private businesses invest enormous resources into the collection, aggre-

gation, and sharing of large amounts of personal data for the tremendous benefit of Big Data. Many facts [section 3] show that Big Data will harm the user's privacy if it is not properly handled (Matturdi et al. 2014). The security and privacy issues which should be concerned in Big Data context include: "1. The personal information of a person when combined with external large data sets, leads to the inference of new facts about that person whereby these facts about the person are sometimes secretive and the person might not want the data owner to know or any person to know about them; 2. Information regarding the users (people) is collected and exploited in order to add value to the business of any organization. This is done by creating insights into their lives which they are unaware of; 3. Another consequence is of Social stratification where a literate person would be taking advantages of the Big data predictive analysis whereas the illiterate/ underprivileged will be worse off if it is high in developing countries where 'Digital Divide' is very much prevalent; 4. Big Data used by law enforcement will increase the chances of certain tagged people to suffer from adverse consequences who never have the ability to defend nor have the knowledge of being discriminated against" (Katal et al. 2013). The field of privacy in Big Data which contains a host of challenges involves interaction with individuals, re-identification attacks, probable and provable results, and economic effects (Jensen 2013; Zhang et al. 2014).

Privacy and Security Issues of Big Data

Data is the most crucial part in *Big Data*. Hence, assuring data security and privacy during either data transition or data storage is the core need of Big Data (Kadhiwala 2017; Zhang et al. 2017). Generally, data is treated as secure, when *confidentiality, integrity, and availability*, i.e., the CIA triad model, are satisfied (Xu and Shi 2016). *PAIN* is another measure to ensure data security and privacy benchmark (Sudarsan et al. 2015; Sun et al. 2011a). Hence, favoring literature studies, the main security and privacy issues of Big Data are confidentiality, integrity,

availability, monitoring and auditing, key management, and data privacy (Cheng et al. 2017).

Data Confidentiality:Research Directions

Confidentiality relates to applying some rules and restrictions to data against illegal disclosure. Confidentiality can be assured by limiting access to the data and also by employing various cryptographic techniques. There are three different ways to ensure confidentiality in typical security mechanism as follows (Sudarsan et al. 2015):

- Data is encrypted during transition and stored as plaintext.
- Authentication is used on stored plaintext data to grant access.
- Data is encrypted when stored and decrypted when in use.

Several data confidentiality techniques exist, and the most significant techniques among those are access control and encryption. Both techniques have been widely investigated (Bertino et al. 2011; Nabeel et al. 2013; Shang et al. 2010), and both are needed to assure data confidentiality. Confidentiality can also be assured using authentication, one of the AAA security concepts (Li et al. 2011; Ulusoy et al. 2014). Authentication is referred as user identity establishment, whereas authorization is used to grant resource access to the authenticated user. Access control refers to enforcing resource access permission for authorized use to authenticated users. Several access control mechanisms such as mandatory access control (MAC) (Balamurugan et al. 2015), RBAC (Balamurugan et al. 2015; Wang et al. 2005), and ABAC (Ruj 2014) can be used to ensure data privacy.

Integrity

Data integrity provides protection against altering of data by an unauthorized user in an unauthorized manner. Hardware errors, user errors, software errors, or intruders are main reasons for data integrity issues (Sudarsan et al. 2015). Salami attacks, data diddling attacks, trust relationship attacks, man in the middle attack, and

session hijacking attacks are most well-known attacks on data integrity (Types of Network Attacks against Confidentiality, Integrity and Availability 2017). Integrity can be maintained using data provenance, data trustworthiness, data loss, and data deduplication. Data provenance is related to information about creation process as well as sources of data through which it is transformed. It is the process to check all states of data from initial state to current state. Debugging, security, and trust models are various applications of data provenance (Azmi 2015; Glavic 2014). Without data provenance information, the user never comes to know from where the data came and what and which transformations have applied to data. This affects the value or originality of data (Alguliyev and Imamverdiyev 2014).

Availability

Data availability ensures that data must be available for use whenever authorized users require it. However, the emergence of cloud computing has narrowed down issues of data availability for Big Data due to high uptime of cloud. Denial of service (DoS) attack, DDoS attack, and SYN flood attack are known attacks to breach data availability; therefore, there is a need of revised solutions (Types of Network Attacks against Confidentiality, Integrity and Availability 2017).

Key Management

Key management and key sharing between users, servers, and data centers are emerging security issues for Big Data. Wen et al. (Jeong and Shin 2016) have explained various approaches for key management such as secret sharing, server-aided approach, and encryption with signature. In key management with RSSS, users do not need to maintain any key on their own, but instead, they have to share secrets among multiple servers. Key can be reconstructed using a minimum defined number of secrets using RSSS.

Data Privacy

Data privacy intends to assure personally identifiable information (PII) should not be shared without informed assent of related data owner (Kabir et al. 2012; Sun et al. 2011b). In some

cases, even though receiving user's consent to use and share the data, the use of PII could be restrained for a specific reason. For example, to develop effective treatment or medicine, the study of patient's medical record is essential. Hence, PII of the patient must be anonymized to protect privacy. In order to ensure data privacy, several encryption techniques and access control mechanisms can be employed.

Need for Data Security: Looking at the Bigger Picture

With the proliferation of data, cyber attacks and data breaches are increasing exponentially (Shen et al. 2017). The growing spate of incidents due to the proliferation and careless handling of data increases the vulnerability of data leakage. With the advent of *cloud*, the data has become more vulnerable to cyber attacks. With the growth of data and its insensitive management, it has come under various sorts of threats that compromise the privacy and security of nations in general and individuals in particular. Recent happenings across the world revealed the vulnerability of data to threats and greater ramifications that followed. Nowadays, no sector is free from data threats and breach. From US presidential elections (where private server of the presidential candidate was hacked) to the recent attack on UIDAI (Unique Identity Development Authority of India) which allots UIN (unique identity number for its citizens), the threat on Big Data is continuing. The incident that shook the conscience of global cyber community was the Ransomware attack that threw bare computers and systems across more than 150 countries at the mercy of cyber thieves. Though some breaches such as "WikiLeaks" and "Panama Papers" expose various irregularities in public domain, these are viewed as great thefts in the literature of data security which put the privacy and security of nations and individuals at stake. Hence there is an imminent need to develop a comprehensive and full proof mechanism to secure the Big Data from unauthorized intrusions.

Key Applications

There are diverse privacy and security concerns in various sectors such as social media, banking sectors, healthcare systems, energy industries, and other online database systems. In this study, we discuss one of the research applications related to Big Data in healthcare or EHD that explains how to enforce privacy and security of Big Data.

Application of Privacy and Security in Electronic Health Data (EHD)

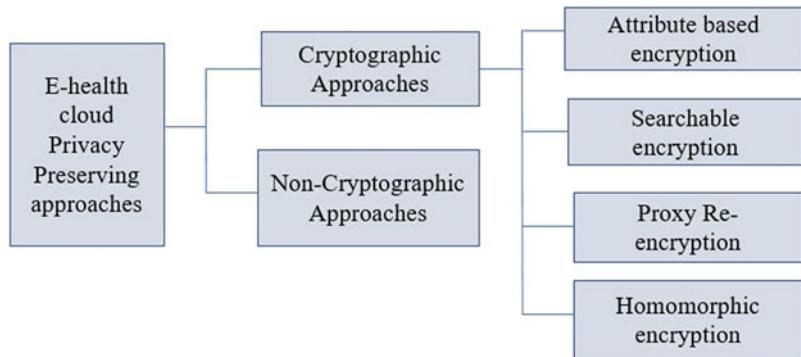
EHD (also known as electronic health records, EHR) is a systematic collection of electronic health information about individual patients or populations (Yi et al. 2013). Such records include a whole range of data including demographics, medical histories, medication and allergies, immunization status, laboratory test results, radiology images, billing information, and all sensitive patient information. According to a national survey, 94% of providers report that their EHR makes records readily available at point of care, 88% report that their EHR produces clinical benefits for the practice, and 75% of providers report that their EHR allows them to deliver better patient care (Clemens et al. 2017). However, the transition from paper-based to EHR systems poses some unique challenges for privacy and confidentiality, security, data integrity, and availability. As *cloud computing* is emerging as a new computing paradigm in healthcare sector (Griebel et al. 2015), it not only facilitates the exchange of electronic medical records among healthcare providers or organizations but also acts as a medical record storage center (Li et al. 2016). *Cloud* services provide the infrastructure to healthcare providers and patients by reducing the cost of storing, processing, and updating information with improved efficiency and quality. As computerized medical records are integrated into one place, data can be accessed from different places by different users, and this increases the risk of invasion of privacy. Since most of the data are sensitive and strictly confidential and stored on a third-party server where the owner doesn't have direct access, it demands serious threats in

terms of data privacy and security (Abbas and Khan 2014; Vimalachandran et al. 2017). This work focuses on identifying the most appropriate method to share private information between multiple healthcare providers in the patient's care team and the patient and their family or carers in the *cloud* environment.

Hence, EHR in healthcare is facing problems with privacy breaches and unauthenticated record access in the recent years, and the most prime one is related to privacy and security of medical data (Abbas and Khan 2014). The issues range from malware attacks that compromise the integrity of systems and privacy of patients to distributed denial of service (DDoS) which can disrupt facilities' ability to provide patient care. In healthcare systems, for instance, cyber attacks like Ransomware can have ramifications beyond financial loss and breach of privacy (Ahmed and Ullah 2017). Earlier this year, hackers broke into the databases of Community Health Systems (CHS), one of the largest hospital groups in the United States, and accessed personal health information, name, address, and personal data including social security numbers from around 4.5 million patients. Hackers from Internet vigilante group Anonymous also targeted several hospitals, launching a DDoS attack on the hospital website as an act of "hacktivism" (AbuKhousa et al. 2012).

Therefore, there is an indispensable need to protect the privacy, security, confidentiality, integrity, and availability of sensitive information pertaining to individuals' data in general. In this context, cybersecurity is utmost required to prevent, detect, and act on unauthorized access to a health system and its information. Therefore, the primary aim is to strengthen the security infrastructure by providing a strong protection mechanism in "e-healthcare" aiming toward patients' confidence and thereby providing security and privacy (Wu et al. 2012) to electronic health data. Therefore, in order to protect the privacy of patient data, access control mechanisms and encryption techniques will be a better solution. Some of the cryptographic approaches are mentioned in Fig. 3. Narayan et al. (2010) proposed an attribute-based infrastructure for the

Security and Privacy in Big Data Environment,
Fig. 3 Taxonomy of the privacy-preserving approaches in the e-Health cloud



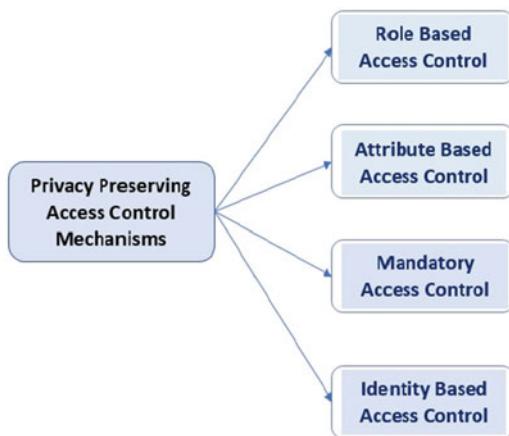
EHR where the patients encrypt their EHR files using the bABE. This approach solves the key management issues by using the users' attribute for data encryption allowing every user to have only one private key for their attribute set for decryption. This approach also allows users to carry and healthcare providers to perform keyword-based searches on the encrypted patients' records without revealing the keywords or partial matches to the cloud system. The keyword search functionality is provided by combining the bABE and the public key encryption with keyword search (PEKS). Ibraimi et al. (2009) proposed a multi-authority scheme for protecting EHD (electronic health data) across different domains. The limitation of the work is that they are not designed for database federations of many medical organizations.

Narayan et al. (2010) proposed a patient-centric cloud-based EHR system by incorporating symmetric key cryptography, public key cryptography, and an attribute-based architecture. This method includes encrypting the patient health data being encrypted by the patient using a symmetric key and also a metadata file which includes a description of the file, attribute-based access policy, and location information encrypted using broadcast CP-ABE and stored in a cloud platform. This method supports direct revocation without re-encryption of data but has a higher computational cost on the patient end where all re-encryption and updating of access policies are handled by the patient side (Narayan et al. 2010). Another downside is that the Trusted Authority can access all the

encrypted files. Barua et al. (2013) used the mechanisms of proxy re-encryption and ABE to develop a more sophisticated cloud-based solution. However it does not require external party for key distribution; it leads to a single point of failure and also creates key escrow while managing all attributes with a single authority. In order to resolve the issue, Li et al. (2013) introduced a cloud-based health record sharing scheme using both KP-ABE and MA-ABE schemes. This solution exerts some level of computational cost on the user side.

In EHR systems as most of the data are strictly confidential and stored in a third-party *cloud*, *access control mechanisms* are equally ineludible and vital as *encryption techniques* (Zhang et al. 2015). Access control is a fundamental security barrier for data privacy in a healthcare information system, which limits who can access and operate the documents in an EHR system (Wang et al. 2005, 2009; Zhang et al. 2014). Some access control mechanisms are mentioned in Fig. 4.

Several access control mechanisms such as RBAC and its variants are currently used, and newer approaches such as ABAC have been proposed in healthcare domain to restrict access to electronic health data (Khalil et al. 2007, Wang et al. 2002). However, both of these have few inherent inadequacies as an individual approach. However, by combining the advantages of both RBAC and ABAC, a new approach can be proposed (Alshehri and Raj 2013). This dual-layer access control model being proposed integrates attributes with roles combining the strength of RBAC and ABAC and thereby aims at assuring



Security and Privacy in Big Data Environment, Fig. 4
Basic access control schemes

two fundamental security properties, confidentiality and integrity, of the sensitive data in the healthcare domain. Several research aspects of privacy and security in EHD are categorized and discussed in this study. However, among several encryption techniques, ABE and combination of several access control mechanisms are profoundly important for enforcing security and privacy to EHD.

Future Research Directions

Privacy and security of Big Data is gaining momentum in research community due to emerging technologies like cloud computing, analytics engines, and social networks. There are a number of open problems and future research perspectives related to privacy and security of Big Data (Cuzzocrea 2014, Wang et al. 2015).

1. Privacy-Preserving Big Data Analytics – Big Data are valuable because they are treasured source of knowledge that is useful for decision making and prediction purposes. It possesses challenging research hurdles, because analytics process huge volumes of Big Data, and hence privacy of target data sets is not preserved yet.
2. Privacy-Preserving Social Network Mining – Social network data are the most reliable

sources of real-life Big Data, with well-known web social networks like Facebook, Twitter and Instagram. Here, data mining is of primary interest, but the need for privacy and security very often limits the real impact of these tasks.

3. Privacy-Preserving Electronic Health Data – Electronic health data stores sensitive and confidential patient information in large datasets. Hence, there is a high need to preserve the privacy and security of stored data sets to protect the confidentiality of patient.
4. Security Issues of (Big) Outsourced Databases – In cloud infrastructures, databases are often outsourced based on the DaaS (Database as a Service) approach. This elevates more problematic security concern as query-processing procedures may easily access sensitive data sets and determine privacy breaches.

Conclusion

Privacy and security of Big Data is gaining prominence nowadays due to its high proliferation and utility as a result of recent developments in information and communication technology (ICT) such as social networking, IoT (Internet of things), Big Data analytics, and cloud computing. Big Data is a treasure trove of knowledge due to its potential for large-scale use and utility across various fields. This paper discusses scopes, challenges, and various techniques employed to secure privacy and security of Big Data. There is ample scope for further research in Big Data security and privacy of which few areas are discussed under the heading “[Future Research Directions](#)” of this write-up. However, in light of existing bottlenecks in data privacy and security, future research directions call for deliberate attention and immense contribution of research scholars and practitioners.

Cross-References

- [Healthcare](#)
- [Security and Privacy in Big Data Environment](#)

References

- Abbas A, Khan SU (2014) A review on the state-of-the-art privacy-preserving approaches in the e-health clouds. *IEEE J Biomed Health Inform* 18: 1431–1441
- AbuKhoua E, Mohamed N, Al-Jaroodi J (2012) E-health cloud: opportunities and challenges. *Futur Internet* 4:621–645
- Ahmed M, Ullah ASB (2017) False data injection attacks in healthcare. In: Australasian Conference on Data Mining. Springer Singapore, Singapore, pp 192–202
- Alguliyev R, Imamverdiyev Y (2014) Big data: big promises for information security. In: Proceedings of IEEE 8th international conference on application of information and communication technologies, pp 1–4
- Alshehri S, Raj RK (2013) Secure access control for health information sharing systems. In: Proceedings of IEEE international conference on healthcare informatics, pp 277–286
- Azmi Z (2015) Opportunities and security challenges of big data. In: Current and emerging trends in cyber operations. Palgrave Macmillan UK, London, pp 181–197
- Balamurugan B, Shivitha NG, Monisha V, Saranya V (2015) Survey of access control models for cloud based real-time applications. In: Proceedings of the international conference on innovation information in computing technologies, 2015. IEEE, pp 1–6
- Barua M, Lu R, Shen X (2013) SPS: spersonal health information sharing with patient-centric access control in cloud computing. In: Proceedings of the IEEE global communications conference (GLOBECOM), 2013 IEEE, pp 647–652
- Bertino E, Ghinita G, Kamra A (2011) Access control for databases: concepts and systems. *Found Trends® Databases* 3:1–148
- Cheng K, Wang L, Shen Y, Wang H, Wang Y, Jiang X, Zhong H (2017) Secure k-NN query on encrypted cloud data with multiple keys. In: IEEE transactions on big data. IEEE. <https://doi.org/10.1109/TBDATA.2017.2707552>
- Clemens S, Alekhya G, Sneha V, Ujwala S, Yazhini C (2017) Impact of electronic health records on long-term care facilities: systematic review. *JMIR Med Inform* 5:e35. <https://doi.org/10.2196/medinform.7958>
- Cuzzocrea A (2014) Privacy and security of big data: current challenges and future research perspectives. In: Proceedings of the first international workshop on privacy and security of big data. ACM, pp 45–47
- Dwork C (2011) Differential privacy. In: Encyclopedia of cryptography and security. Springer-Verlag Berlin, Heidelberg, pp 338–340
- Fang W, Wen XZ, Zheng Y, Zhou M (2017) A survey of big data security and privacy preserving. *IETE Tech Rev* 34:544–560
- Gantz J, Reinsel D (2012) The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze Futur 2007: 1–16
- Glavic B (2014) Big data provenance: challenges and implications for benchmarking. In: Specifying big data benchmarks. Springer Berlin Heidelberg, pp 72–80
- Griebel L et al (2015) A scoping review of cloud computing in healthcare. *BMC Med Inform Decis Mak* 15:17. <https://doi.org/10.1186/s12911-015-0145-7>
- Huang J, Peng M, Wang H, Cao J, Gao W, Zhang X (2017) A probabilistic method for emerging topic tracking in microblog stream. *World Wide Web* 20(2): 325–350
- Ibraimi L, Asim M, Petković M (2009) Secure management of personal health records by applying attribute-based encryption. In: Proceedings of 2009 6th international workshop on wearable micro and nano technologies for personalized health (pHealth). IEEE, pp 71–74
- Jensen M (2013) Challenges of privacy protection in big data analytics. In: Proceedings of 2013 IEEE international congress on big data (BigData Congress). IEEE, pp 235–238
- Jeong Y-S, Shin S-S (2016) An efficient authentication scheme to protect user privacy in seamless big data services. *Wirel Pers Commun* 86:7–19
- Kabir ME, Wang H, Bertino E (2012) A role-involved purpose-based access control model. *Inf Syst Front* 14:809–822
- Kadhiwala NJaB (2017) Big data security and privacy issues – a survey. In: Proceedings of the international conference on innovations in power and advanced computing technologies (i-PACT). pp 1–5. <https://doi.org/10.1109/IPACT.2017.8245064>
- Katal A, Wazid M, Goudar R (2013) Big data: issues, challenges, tools and good practices. In: Proceedings of 2013 sixth international conference on contemporary computing (IC3). IEEE, pp 404–409
- Khalil F, Wang H, Li J (2007) Integrating markov model with clustering for predicting web page accesses. In: Proceeding of the 13th Australasian world wide web conference, pp 63–74
- Li N, Li T, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: Proceeding of the IEEE 23rd international conference on data engineering (ICDE 2007). IEEE, pp 106–115
- Li M, Sun X, Wang H, Zhang Y, Zhang J (2011) Privacy-aware access control with trust management in web service. *World Wide Web* 14:407–430
- Li M, Yu S, Zheng Y, Ren K, Lou W (2013) Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Trans Parallel Distrib Syst* 24:131–143
- Li P, Guo S, Miyazaki T, Xie M, Hu J, Zhuang W (2016) Privacy-preserving access to big data in the cloud. *IEEE Cloud Comput* 3:34–42
- Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) L-diversity: privacy beyond k-anonymity. *ACM Trans Knowl Discov Data* 1:3

- Matturdi B, Xianwei Z, Shuai L, Fuhong L (2014) Big data security and privacy: a review. *China Commun* 11:135–145
- McCune JC (1998) Data, data, everywhere. *Manag Rev* 87:10
- Nabeel M, Shang N, Bertino E (2013) Privacy preserving policy-based content sharing in public clouds. *IEEE Trans Knowl Data Eng* 25:2602–2614
- Narayan S, Gagné M, Safavi-Naini R (2010) Privacy preserving EHR system using attribute-based infrastructure. In: Proceedings of the 2010 ACM workshop on cloud computing security workshop. ACM, pp 47–52
- Ruij S (2014) Attribute based access control in clouds: a survey. In: Proceedings of the 2014 international conference on signal processing and communications (SPCOM). IEEE, pp 1–6
- Shang N, Nabeel M, Paci F, Bertino E (2010) A privacy-preserving approach to policy-based content dissemination. In: Proceedings of 2010 IEEE 26th international conference on data engineering (ICDE). IEEE, pp 944–955
- Shen Y, Zhang T, Wang Y, Wang H, Jiang X (2017) MicroThings: a generic iot architecture for flexible data aggregation and scalable service cooperation. *IEEE Commun Mag* 55:86–93
- Shu J, Jia X, Yang K and Wang H (2018) Privacy-preserving task recommendation services for crowdsourcing. *IEEE Trans Serv Comput* 1(99):1–1
- Sudarsan SD, Jetley RP, Ramaswamy S (2015) Security and privacy of big data. In: Big data. Springer India, New Delhi, pp 121–136
- Sun X, Wang H, Li J, Pei J (2011a) Publishing anonymous survey rating data. *Data Min Knowl Disc* 23:379–406
- Sun X, Wang H, Li J, Zhang Y (2011b) Injecting purpose and trust into data anonymisation. *Comput Secur* 30:332–345
- Sweeney L (2002) K-anonymity: a model for protecting privacy. *Int J Uncertainty Fuzziness Knowledge Based Syst* 10:557–570
- Types of Network Attacks against Confidentiality, Integrity and Availability (2017) <http://www.omnisecu.com/ccna-security/types-of-network-attacks.php>. Accessed 23 Jan 2017
- Ulusoy H, Kantarcioğlu M, Pattuk E, Hamlen K (2014) Vigiles: fine-grained access control for mapreduce systems. In: Proceedings of 2014 IEEE international congress on big data (BigData Congress). IEEE, pp 40–47
- Venkatram K, Geetha MA (2017) Review on big data & analytics – concepts, philosophy, process and applications. *Cybern Inf Technol* 17:3–27
- Vimalachandran P, Wang H, Zhang Y, Zhuo G, Kuang H (2017) Cryptographic access control in electronic health record systems: a security implication. In: Proceedings of the international conference on web information systems engineering. Springer, pp 540–549
- Wang H, Cao J, Zhang Y (2002) Ticket-based service access scheme for mobile users. *Aust Comput Sci Commun* 24(1):285–292
- Wang H, Cao J, Zhang Y (2005) A flexible payment scheme and its role-based access control. *IEEE Trans Knowl Data Eng* 17:425–436
- Wang H, Zhang Y, Cao J (2009) Effective collaboration with information sharing in virtual universities. *IEEE Trans Knowl Data Eng* 21(6):840–853
- Wang H, Jiang X, Kambourakis G (2015) Special issue on security, privacy and trust in network-based big data. *Inf Sci Int J* 318:48–50
- Wu R, Ahn G-J, Hu H (2012) Secure sharing of electronic health records in clouds. In: Proceedings of 2012 8th international conference on collaborative computing: networking, applications and worksharing (CollaborateCom). IEEE, pp 711–718
- Xu L, Shi W (2016) Security theories and practices for big data. In: Big data concepts, theories, and applications. Springer International Publishing, Cham, pp 157–192
- Yi X, Miao Y, Bertino E, Willemson J (2013) Multiparty privacy protection for electronic health records. In: Proceedings of the global communications conference (GLOBECOM), 2013 IEEE. IEEE, pp 2730–2735
- Zhang J, Tao X, Wang H (2014) Outlier detection from large distributed databases. *World Wide Web* 17: 539–568
- Zhang Y, Shen Y, Wang H, Yong J, Jiang X (2015) On secure wireless communications for IoT under eavesdropper collusion. *IEEE Trans Autom Sci Eng* 13(3):1281–1293. July 2016
- Zhang J, Li H, Liu X, Luo Y, Chen F, Wang H, Chang L (2017) On efficient and robust anonymization for privacy protection on massive streaming categorical information. *IEEE Trans Dependable Secure Comput* 14(5):507–520

S

Self-Supervised Relation Extraction

► Distant Supervision from Knowledge Graphs

Semantic Computing

► Automated Reasoning

Semantic Data Compression

► RDF Compression

Semantic Interlinking

Gianluca Demartini

The University of Queensland, St. Lucia, QLD, Australia

Definitions

Semantic interlinking is defined as the establishment of links and relations between multiple structured datasets.

Overview

Motivation

The exponential growth of data is becoming pervasive across different areas of business and science. Despite its wide availability in large amounts, data is typically stored in standalone silos where different datasets are represented using different formats, stored and indexed within different system architectures, and maintained following different business processes. For example, in certain organizations it is possible to encounter customer databases, technical reports, product images, and other datasets that need to be used in conjunction. Such data integration problems are a long-standing open research challenge in the data management area. The recent rise of big data with its volume and variety dimensions has magnified already existing issues.

Similar challenges are also often present in Open Data where datasets are published and made freely available (typically by governmental organizations) without paying much attention at data quality issues such as the lack of appropriate data format usage and the provision of datasets out of their context.

These situations make data integration an open challenge. *Semantic data interlinking* can be generally defined as the establishment of links and relations between multiple structured datasets. This entry presents an overview of semantic techniques for the interlinking of big data.

Linked Open Data

An important application domain of semantic interlinking is Linked Open Data (LOD) where the goal is to publishing data openly, in a structured format, and interlinked together (Bizer et al. 2007). The aim of the LOD initiative is to fix the challenges that open data comes with, by means of using a common data representation model and by interlinking datasets together.

Tim Berners-Lee suggested a 5-star model for LOD (<http://5stardata.info>) where he claims that data shared on the Web should ideally be (1) available under an open license, (2) provided as structured data, (3) published using nonproprietary formats, (4) described using unique resource identifiers (URIs), and (5) linked to other data to provide context.

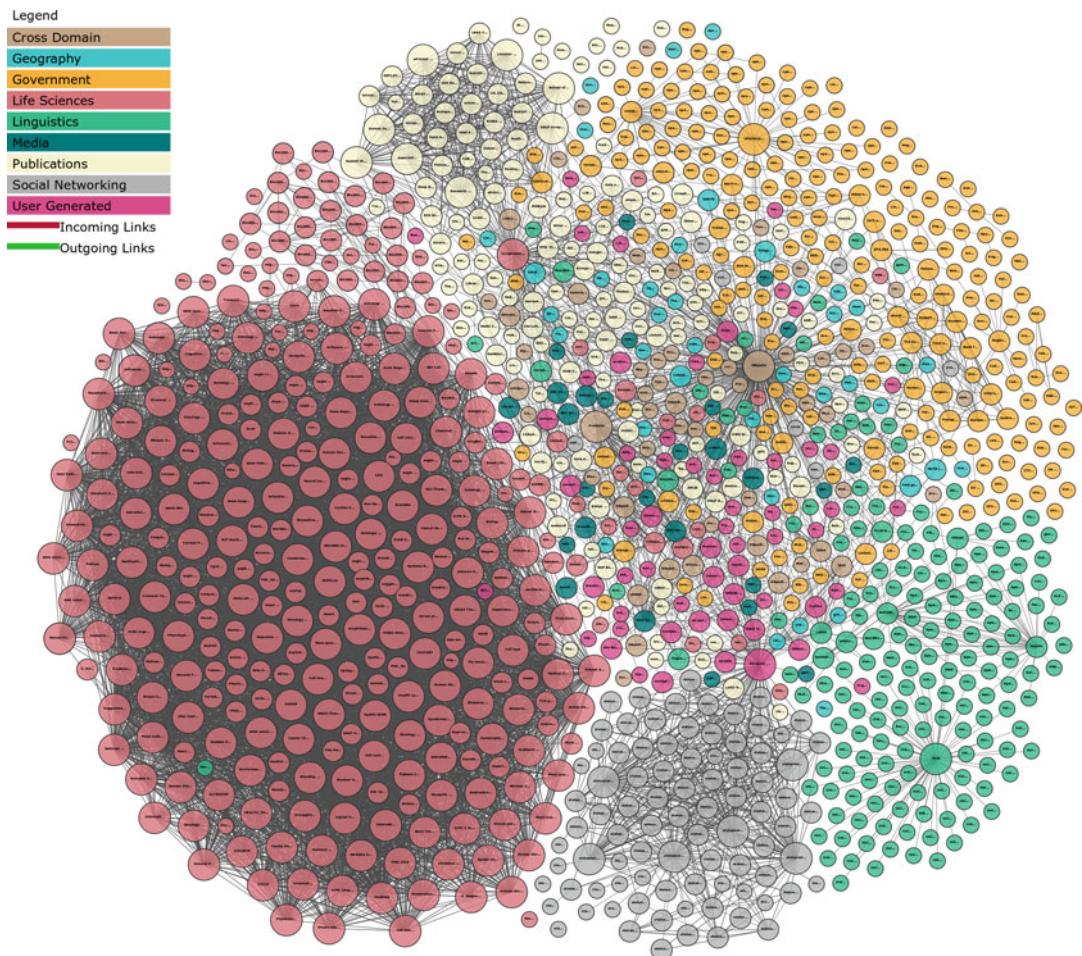
Existing datasets that have been published following such principles are depicted in the LOD cloud (see Fig. 1) where each bubble represents a dataset following LOD principles and edges between nodes represent links across items described in the datasets.

Other than domain-specific datasets (many of which are in the biomedical field), popular datasets in the LOD cloud include DBpedia (Auer et al. 2007) and Wikidata (Vrandečić and Krötzsch 2014).

DBpedia is a structured dataset automatically extracted from Wikipedia infoboxes that contains factual statements about notable entities described in Wikipedia. On the other hand, Wikidata is a crowdsourcing effort aiming at creating a knowledge base of facts. The original motivation to start Wikidata was to supply a central repository for the structured information to be displayed in Wikipedia infoboxes.

Key Research Findings

Research in the area of semantic interlinking focuses on different problems: First, given two datasets to be interlinked, there is the need to align the underlying ontologies used to describe the instances present in the datasets; Next, there is the need to align the instances described in the two datasets, that is, to identify which entities



Semantic Interlinking, Fig. 1 Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. <http://lod-cloud.net/>

referred to are the same across the datasets. This section first discusses these two problems and present some methods proposed in the literature designed to address them. It then presents the more specific problem of entity linking, that is, performing semantic interlinking between a document and a structured dataset. Finally, it introduces more recent human-in-the-loop approaches.

Semantic Schema Alignment

The problem of schema alignment (also known as ontology matching Shvaiko and Euzenat 2013) consists in identifying which schema elements of a datasets are equivalent to those of a different

dataset. This is a classic problem in the area of semantic interlinking for which a large number of approaches have been proposed.

For example, Jain et al. (2010) focus on methods to find alignments between ontologies used by different LOD datasets. The approach they propose is based on using the Wikipedia category hierarchy to bootstrap the schema alignment process. Similarly, Parundekar et al. (2010) align different ontologies used in different LOD datasets. Their approach is based on existing equivalence statements at the instance level on which they reason about possible schema alignments. To evaluate such methods, standard benchmarks exist. The most popular and commonly used ones

have been created in the context of the Ontology Evaluation Alignment Initiative (Euzenat et al. 2011).

Semantic Record Linkage

Record linkage is defined as the identification of the same real-world entity mentioned across different datasets. In the semantic interlinking domain, this translates into identifying which instances in two LOD datasets refer to the same object.

Approaches for this problem include, for example, Rong et al. (2012) who look at this as a binary classification problem (i.e., a candidate pair of instances matches or not) and solve it using standard supervised machine learning models.

The main challenge of record linkage is scalability: When aiming at identifying duplicates across two datasets, it would be necessary to perform a quadratic number of comparisons to check every possible pair of instances. In a big data context, where the volume of data is prohibitive, performing all possible comparisons is not a scalable option. To deal with this challenge, a number of indexing techniques to make more efficient comparisons have been proposed (Christen 2012). Another common approach to deal with this is *blocking* where the idea is to first use computationally inexpensive methods to create groups of similar objects based on approximate methods (e.g., by means of clustering) and then to perform all possible comparisons using a computationally expensive similarity measure only for the members of a group thus removing a large number of false positives (Bilenko et al. 2006; Papadakis et al. 2013).

Entity Linking

A third problem related to the two main semantic interlinking problems described above is that of *entity linking* (Rao et al. 2013). This is defined as uniquely disambiguating an entity mentioned in a textual document by linking it to a background knowledge graph (Shen et al. 2015). For example, given a document mentioning the entity "Tom Cruise," the goal is to create a link from it to the URI of an instance of a LOD dataset (e.g., DBpedia).

Common approaches for entity linking include, for example, graph-based approaches (Moro et al. 2014) able to look at the coherence of linking decisions based on sub-graph density.

When run at scale, entity linking can benefit from collection features like, for example, entity frequency as an indicator of linking accuracy (Lin et al. 2012). That is, an entity which appears several times in a coherent document collection is likely to be referring to the same concept. It is also easier to use contextual information (e.g., all the sentences where an entity appears) to better decide about which instance of the knowledge graph to link to.

Human-in-the-Loop Semantic Interlinking

A more recent family of approaches to semantic interlinking makes use of the *crowdsourcing* methodology. This implies the development of *human-in-the-loop systems* that leverage machine-based computation to scale interlinking to large datasets but make also use of crowdsourcing to solve difficult cases where humans outperform machine-based algorithms.

An early approach to crowd-based schema alignment was presented by Sarasua et al. (2012) where they compared a human-in-the-loop approach with purely machine-based schema alignment methods showing significant improvements in alignment effectiveness.

For the problem of entity linking, human-in-the-loop solutions include ZenCrowd by Demartini et al. (2012) where authors propose to, given a document and background LOD dataset, collect entity linking decisions from algorithms and from a crowdsourcing platforms. Then, they propose a factor graph model to effectively combine the machine-based and human-based decisions. Again, experimental results show that such combined approach results in better quality entity linking.

A similar human-in-the-loop approach has been proposed by Demartini et al. (2013) for semantic record linkage where the crowdsourcing step is used last after the machine-based blocking and alignment methods described above. For the same problem of semantic record linkage, Wang et al. (2012) investigated the use of grouping

record linkage crowdsourcing tasks together showing how crowd workers perform better as compared to when they are presented with individual record linkage tasks.

Examples of Application

Potential applications where semantic interlinking can provide benefits include information extraction, information retrieval, and knowledge base population.

Applications domains where LOD datasets and semantic interlinking have been used successfully include cultural heritage (Knoblock et al. 2017), manufacturing (Petersen et al. 2017), smart cities (Egami et al. 2016), and others.

Future Directions for Research

As this entry has shown, research in the area of semantic interlinking has been focusing on a number of diverse problems (i.e., semantic schema alignment, semantic record linkage, and entity linking). The current focus of the research community is on the scalability of such approaches to large heterogeneous datasets (e.g., by means of blocking and indexing techniques) and on improving interlinking accuracy by means of human-in-the-loop systems.

Future work should be looking at issues like, for example, semantic interlinking for dynamic LOD datasets (i.e., datasets that are not considered static but rather evolving over time) for which already existing interlinking decisions may need to be updated as ontologies used by the LOD datasets may change (Kuhn et al. 2017).

Cross-References

- ▶ [Linked data management](#)
- ▶ [Record Linkage](#)
- ▶ [Schema Mapping](#)

References

Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z (2007) Dbpedia: a nucleus for a web of open

- data. In: The semantic web, 6th international semantic web conference, 2nd Asian semantic web conference, ISWC 2007 + ASWC 2007, Busan, Korea, 11–15 Nov 2007. Springer, Berlin, pp 722–735
- Bilenko M, Kamath B, Mooney RJ (2006) Adaptive blocking: learning to scale up record linkage. In: Sixth international conference on data mining (ICDM'06), pp 87–96. <https://doi.org/10.1109/ICDM.2006.13>
- Bizer C, Heath T, Ayers D, Raimond Y (2007) Interlinking open data on the web. In: Demonstrations track, 4th European semantic web conference, Innsbruck
- Christen P (2012) A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans Knowl Data Eng 24(9):1537–1555. <https://doi.org/10.1109/TKDE.2011.127>
- Demartini G, Difallah DE, Cudré-Mauroux P (2012) Zen-crowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In: Proceedings of the 21st international conference on world wide web. ACM, pp 469–478
- Demartini G, Difallah DE, Cudré-Mauroux P (2013) Large-scale linked data integration using probabilistic reasoning and crowdsourcing. VLDB J 22(5):665–687
- Egami S, Kawamura T, Ohsuga A (2016) Building urban LOD for solving illegally parked bicycles in Tokyo. In: Proceedings 15th international semantic web conference, part II, the semantic web – ISWC 2016, Kobe, 17–21 Oct 2016, pp 291–307. https://doi.org/10.1007/978-3-319-46547-0_28
- Euzenat J, Meilicke C, Stuckenschmidt H, Shvaiko P, Trojahn C (2011) Ontology alignment evaluation initiative: six years of experience. In: Spaccapietra S (ed) Journal on data semantics XV. Springer, Berlin, pp 158–192
- Jain P, Hitzler P, Sheth AP, Verma K, Yeh PZ (2010) Ontology alignment for linked open data. Springer, Berlin/Heidelberg, pp 402–417. https://doi.org/10.1007/978-3-642-17746-0_26
- Knoblock CA, Szekely PA, Fink EE, Degler D, Newbury D, Sanderson R, Blanch K, Snyder S, Chheda N, Jain N, Krishna RR, Sreekanth NB, Yao Y (2017) Lessons learned in building linked data for the American art collaborative. In: Proceedings of the 16th international semantic web conference, part II, the semantic web – ISWC 2017, Vienna, 21–25 Oct 2017, pp 263–279. https://doi.org/10.1007/978-3-319-68204-4_26
- Kuhn T, Willighagen E, Evelo C, Queralt-Rosinach N, Centeno E, Furlong LI (2017) Reliable granular references to changing linked data. Springer International Publishing, Cham, pp 436–451. https://doi.org/10.1007/978-3-319-68288-4_26
- Lin T, Mausam, Etzioni O (2012) Entity linking at web scale. In: Proceedings of the joint workshop on automatic knowledge base construction and web-scale knowledge extraction, association for computational linguistics, AKBC-WEKEX '12, Stroudsburg, pp 84–88. <http://dl.acm.org/citation.cfm?id=2391200.2391216>
- Moro A, Raganato A, Navigli R (2014) Entity linking meets word sense disambiguation: a unified approach. Trans Assoc Comput Linguist 2:231–244

- Papadakis G, Ioannou E, Palpanas T, Niederee C, Nejdl W (2013) A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans Knowl Data Eng* 25(12):2665–2682
- Parundekar R, Knoblock CA, Ambite JL (2010) Linking and building ontologies of linked data. Springer, Berlin/Heidelberg, pp 598–614. https://doi.org/10.1007/978-3-642-17746-0_38
- Petersen N, Halilaj L, Grangé-González I, Lohmann S, Lange C, Auer S (2017) Realizing an RDF-based information model for a manufacturing company – a case study. In: Proceedings of the 16th international semantic web conference, part II, the semantic web – ISWC 2017, Vienna, 21–25 Oct 2017, pp 350–366. https://doi.org/10.1007/978-3-319-68204-4_31
- Rao D, McNamee P, Dredze M (2013) Entity linking: finding extracted entities in a knowledge base. Springer, Berlin/Heidelberg, pp 93–115. https://doi.org/10.1007/978-3-642-28569-1_5
- Rong S, Niu X, Xiang EW, Wang H, Yang Q, Yu Y (2012) A machine learning approach for instance matching based on similarity metrics. Springer, Berlin/Heidelberg, pp 460–475. https://doi.org/10.1007/978-3-642-35176-1_29
- Sarasua C, Simperl E, Noy NF (2012) Crowdmap: crowdsourcing ontology alignment with microtasks. In: International semantic web conference. Springer, pp 525–541
- Shen W, Wang J, Han J (2015) Entity linking with a knowledge base: issues, techniques, and solutions. *IEEE Trans Knowl Data Eng* 27(2):443–460
- Shvaiko P, Euzenat J (2013) Ontology matching: state of the art and future challenges. *IEEE Trans Knowl Data Eng* 25(1):158–176. <https://doi.org/10.1109/TKDE.2011.253>
- Vrandečić D, Krötzsch M (2014) Wikidata: a free collaborative knowledgebase. *Commun ACM* 57(10):78–85
- Wang J, Kraska T, Franklin MJ, Feng J (2012) Crowder: crowdsourcing entity resolution. *Proc VLDB Endow* 5(11):1483–1494

Semantic Search

Philippe Cudre-Mauroux
eXascale Infolab, University of Fribourg,
Fribourg, Switzerland

Definitions

Semantic Search regroups a set of techniques designed to improve traditional document or knowledge base search. Semantic Search aims at better grasping the context and the semantics of the user query and/or of the indexed content by leveraging

natural language processing, Semantic Web, and machine learning techniques to retrieve more relevant results from a search engine.

Overview

Semantic Search is an umbrella term regrouping various techniques for retrieving more relevant content from a search engine. Traditional search techniques focus on ranking documents based on a set of keywords appearing both in the user's query and in the indexed content. Semantic Search, instead, attempts to better grasp the semantics (i.e., meaning) and the context of the user query and/or of the indexed content in order to retrieve more meaningful results.

Semantic Search techniques can be broadly categorized into two main groups depending on the target content:

- techniques improving the relevance of classical search engines where the query consists of natural language text (e.g., a list of keywords) and results are a ranked list of documents (e.g., webpages);
- techniques retrieving semi-structured data (e.g., entities or RDF triples) from a knowledge base (e.g., a knowledge graph or an ontology) given a user query formulated either as natural language text or using a declarative query language like SPARQL.

Those two groups are described in more detail in the following section. For each group, a wide variety of techniques have been proposed, ranging from natural language processing (to better grasp the contents of the query and data) to Semantic Web (to guide the search process leveraging declarative artifacts like ontologies) and machine learning (typically to learn models from large quantities of data).

Main Approaches

We give below an overview of the various techniques that have been proposed in the context of Semantic Search for improving document search

as well as knowledge base search. A number of surveys delve into more detail in this context: Mangold (2007) focuses on natural language queries on RDF knowledge bases or ontologies. Madhu et al. (2011) and Mäkelä (2005) are two brief surveys covering both topics. Bast et al. (2016) is an extensive survey covering Semantic Search in its broadest sense.

Document Search

Classical search engines take as input a user query formulated as a list of keywords and return as output a ranked list of documents relevant to those keywords. A number of Semantic Search methods have been suggested in that context.

Natural language processing (NLP) techniques have long been applied to better grasp the semantics of the query or documents. Often, Part-of-Speech (POS) tagging is first applied on the textual content in order to assign grammatical tags (such as *noun*, *conjunction*, or *verb*) to individual words. Such assignment is highly accurate for well-formed sentences (Manning 2011) but much more challenging for short texts such as queries (Hua et al. 2015). POS tags can then be used to better discriminate textual keywords, for example, for named-entity recognition (NER), where the task is to identify which keywords correspond to real-world entities, or for co-reference resolution, where the task is to identify all keywords referring to the same entity in the text. Sentence parsing takes NLP analyses to the next level by aiming at capturing the overall structure of sentences, typically through a dependency parse tree.

NLP methods are often combined with lexical resources or third-party sources to retrieve more relevant results. The main idea in this context is to identify entities in the textual query or content and to match them to their counterpart in a third-party resource to improve the search results. Voorhees (1993) proposed an early approach in the sense that leverages WordNet to disambiguate word senses and hence improve search results. Pehcevski et al. (2008) analyze the structure of Wikipedia to better rank relevant entities in response to a search request. Kaptein

et al. (2010) use Wikipedia to better characterize and identify entities when searching for entities in document collections, while Schuhmacher et al. (2015) combine different features from the documents, the entity mentions, and Wikipedia using a learning-to-rank approach to improve the search results.

Conceptually similar approaches have been proposed in the context of the Semantic Web, by leveraging the structure or contents of a knowledge base to better grasp the context of queries or entities appearing in textual documents. Tran et al. (2007), for instance, propose an ontology-based interpretation of natural language queries for Semantic Search. The authors translate a keyword query into a description logic, conjunctive query that can then be evaluated with respect to an underlying knowledge base. Schuhmacher and Ponzetto (2013) exploit entities and semantic relations from the DBpedia knowledge base to cluster the results of a search engine into more meaningful groups. Prokofyev et al. (2015) leverage an ontology to better resolve co-references in textual documents for Semantic Search tasks.

Machine learning techniques are often used for Semantic Search, to power some of the approaches described above but also to capture the context and semantics of the words or entities appearing in documents. One of the main intuitions in this context is that words that occur in similar contexts are likely to be semantically similar. Early approaches leveraging this observation built high-dimensional matrices capturing the co-occurrence of words in a certain context (e.g., within a window of a few words) and hence the similarity between words (Lund and Burgess 1996). Each word is in that case represented by a sparse vector in a high-dimensional space. Lower-dimensional embeddings can then be created by applying standard matrix factorization techniques like principal component analysis.

More recently, Mikolov et al. (2013) suggested a new word embedding technique to generate dense vector representations of words

efficiently. The method works by maximizing the co-occurrence probability of words appearing within a certain context window using a relatively simple neural network. This opened the door to numerous applications, by efficiently generating vector representations of words from large text corpora and feeding them into subsequent machine learning models. Approaches to improve entity recognition (Siencnik 2015), web search query expansion (Grbovic et al. 2015), or web search ranking (Nalisnick et al. 2016) have, for example, been explored in the context of Semantic Search.

Knowledge Base Search

A number of Semantic Search approaches target large and declarative knowledge bases (a.k.a ontologies or knowledge graphs) instead of document collections. Such knowledge bases can be expressed in many different ways that are typically derived from Semantic Web standards such as RDF or OWL. Google’s Knowledge Graph, DBpedia (Bizer et al. 2009), Yago (Rebele et al. 2016), or Wikidata (Vrandecic and Krötzsch 2014) are well-known examples of that trend. Users can express their queries through two main modalities in this case: either as structured (e.g., SPARQL) queries or as natural language (e.g., keyword) queries.

Horrocks and Tessaris (2002) introduced an early formal approach to answer structured queries posed against ontologies. Their algorithm returns sound and complete results to conjunctive queries leveraging reasoning techniques and description logics. Stojanovic et al. (2003) present a method to rank results in ontology-based search. The authors consider conjunctive queries and combine logical inference with an analysis of the contents of the knowledge base to retrieve more relevant results. Maedche et al. (2003) introduce a meta-ontology and a registry to improve search queries targeting ontologies. Their solution leverages WordNet to match entities appearing in the ontology to lexical entries and to guide the search process.

Pound et al. (2010) introduce the ad hoc object retrieval task for searching for resources (e.g., entities, types, or relations) over knowledge bases

using natural language queries. The authors also propose a baseline technique for answering such queries based on term frequencies as well as an evaluation methodology. Tonon et al. (2012) propose an improved search technique for ad hoc object retrieval exploiting sequentially an inverted index to answer keyword queries and a graph database to improve the search effectiveness by automatically generating declarative queries over an RDF graph.

Hybrid approaches leveraging both textual and structured contents have also been suggested. Rocha et al. (2004), for instance, combine a traditional search engine with graph exploration techniques to answer keyword queries on an ontology. Zhang et al. (2005) suggest a new model to search semantic portals, where both documents and structured data are available. Their method is based on creating textual representations for all entities in the structured repository such that they can also be indexed and searched through classical information retrieval techniques.

Word embedding techniques (see above) have also been adapted to power Semantic Search on knowledge bases. RDF2Vec (Ristoski and Paulheim 2016), for instance, learns vectorial representations of entities in RDF graphs. Wang et al. (2017) provide a survey of embedding approaches for knowledge bases and classify the models into two main families: translation-based techniques, which interpret relations in the knowledge base as a translation vector between the two entities connected by the relation, and semantic matching models, which exploit similarity-based scoring functions to create the embeddings.

Systems

Leading industrial search engines, such as Bing, Yandex, or Google, all implement Semantic Search in one way or another but typically do not describe in detail the techniques they leverage. A number of Semantic Search systems have been described or open-sourced, however, and are summarized below.

TAP (Guha et al. 2003) is an early Semantic Search framework. TAP focuses on entity search queries expressed as keywords and augments traditional results (documents) with semi-structured data returned from a knowledge base. The knowledge base is also used to better filter and sort the list of returned documents in that context.

A number of Semantic Search systems focusing on RDF and ontologies have been proposed. Swoogle (Ding et al. 2004) offers semantic search over a knowledge base represented in RDF thanks to an inverted index and a database storing metadata about all entities. SWSE (Hogan et al. 2011) follows the typical architecture of a search engine but operates on RDF data also. SWSE results are ranked by running a classical PageRank algorithm on a graph connecting the URIs appearing in the RDF triple to their source on the web.

SemSearch (Lei et al. 2006) is a search engine for the Semantic Web that hides the complexity of the underlying RDF data to the user. SemSearch accepts keyword queries from the user, translates the user queries into formal queries by exploiting the labels of the entities in the knowledge base, runs the resulting query in the knowledge base, and finally ranks the results by taking into account the number of keywords the search results satisfy. Sindice (Oren et al. 2008) is a Semantic Search engine and look-up service that focuses on scaling to very large quantities of semi-structured data. It supports keyword and URI-based search as well as structured queries.

SHOE (Heflin and Hendler 2000) is an early Semantic Search system collecting semi-structured annotations from the web and storing them in a knowledge base. It offers a GUI to formulate ontology-based structured queries to find webpages. The Watson system (d'Aquin and Motta 2011) works similarly by collecting, analyzing, and giving access to ontologies and semantic data available online. It supports both keyword and SPARQL search queries.

SCORE (Sheth et al. 2002) is a platform supporting the creation and maintenance of large knowledge bases. It supports ontology-driven Semantic Search capabilities by extracting facts

and metadata from web sources via text mining techniques.

Nordlys (Hasibi et al. 2017) is an open-source toolkit for Semantic Search. The toolkit supports a number of features including detecting entities in natural language queries, cataloging entities from a knowledge base (by default DBpedia), interlinking entities, and retrieving entities.

Schema.org (Mika 2015), finally, is not a system per se but rather a standardization effort founded by leading search engines (including Google, Microsoft, Yahoo, and Yandex). Its mission is to create and promote schemas to embed semi-structured data in web documents (and beyond) in order to facilitate Semantic Search capabilities online.

Cross-References

- ▶ [Knowledge Graph Embeddings](#)
- ▶ [Reasoning at Scale](#)
- ▶ [Semantic Interlinking](#)

References

- Bast H, Buchhold B, Haussmann E (2016) Semantic search on text and knowledge bases. *Found Trends Infor Retr* 10(2–3):119–271. <http://dx.doi.org/10.1561/1500000032>
- Bizer C, Lehmann J, Kobilarov G, Auer S, Becker C, Cyganiak R, Hellmann S (2009) Dbpedia – a crystallization point for the web of data. *J Web Sem* 7(3):154–165. <https://doi.org/10.1016/j.websem.2009.07.002>
- d'Aquin M, Motta E (2011) Watson, more than a semantic web search engine. *Semant Web* 2(1):55–63. <http://dl.acm.org/citation.cfm?id=2019470.2019476>
- Ding L, Finin T, Joshi A, Pan R, Cost RS, Peng Y, Reddivari P, Doshi V, Sachs J (2004) Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the thirteenth ACM international conference on information and knowledge management, CIKM'04. ACM, New York, pp 652–659. <http://doi.acm.org/10.1145/1031171.1031289>
- Grbovic M, Djuric N, Radosavljevic V, Silvestri F, Bhamidipati N (2015) Context- and content-aware embeddings for query rewriting in sponsored search. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, SIGIR'15. ACM, New York, pp 383–392. <http://doi.acm.org/10.1145/2766462.2767709>

- Guha R, McCool R, Miller E (2003) Semantic search. In: Proceedings of the 12th international conference on world wide web, WWW'03. ACM, New York, pp 700–709. [https://doi.acm.org/10.1145/775152.775250](http://doi.acm.org/10.1145/775152.775250)
- Hasibi F, Balog K, Garigliotti D, Zhang S (2017) Nordlys: a toolkit for entity-oriented and semantic search. In: Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval, SIGIR'17. ACM, New York, pp 1289–1292. [https://doi.acm.org/10.1145/3077136.3084149](http://doi.acm.org/10.1145/3077136.3084149)
- Heflin J, Hendler J (2000) Searching the web with shoe. In: AAAI workshop on artificial intelligence for web search, pp 35–40
- Hogan A, Harth A, Umbrich J, Kinsella S, Polleres A, Decker S (2011) Searching and browsing linked data with swse: the semantic web search engine. Web Sem Sci Serv Agents World Wide Web 9(4):365–401. <http://www.sciencedirect.com/science/article/pii/S1570826811000473>. jWS special issue on Semantic Search
- Horrocks I, Tessaris S (2002) Querying the semantic web: a formal approach. In: Horrocks I, Hendler J (eds) The semantic web—ISWC 2002. Springer, Berlin/Heidelberg, pp 177–191
- Hua W, Wang Z, Wang H, Zheng K, Zhou X (2015) Short text understanding through lexical-semantic analysis. In: 2015 IEEE 31st international conference on data engineering, pp 495–506. <https://doi.org/10.1109/ICDE.2015.7113309>
- Kaptein R, Serdyukov P, de Vries AP, Kamps J (2010) Entity ranking using wikipedia as a pivot. In: Proceedings of the 19th ACM conference on information and knowledge management, CIKM 2010, Toronto, 26–30 Oct, pp 69–78. <http://doi.acm.org/10.1145/1871437.1871451>
- Lei Y, Uren VS, Motta E (2006) Semsearch: a search engine for the semantic web. In: Proceedings of the 15th international conference on managing knowledge in a world of networks, EKAW 2006, Podebrady, 2–6 Oct 2006, pp 238–245. https://doi.org/10.1007/11891451_22
- Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behav Res Methods Instrum Comput 28(2):203–208. <https://doi.org/10.3758/BF03204766>
- Madhu G, Govardhan A, Rajinikanth TV (2011) Intelligent semantic web search engines: a brief survey. CoRR abs/1102.0831. <http://arxiv.org/abs/1102.0831>, 1102.0831
- Maedche A, Motik B, Stojanovic L, Studer R, Volz R (2003) An infrastructure for searching, reusing and evolving distributed ontologies. In: Proceedings of the 12th international conference on world wide web, WWW'03. ACM, New York, pp 439–448. [https://doi.acm.org/10.1145/775152.775215](http://doi.acm.org/10.1145/775152.775215)
- Mäkelä E (2005) Survey of semantic search research. <https://seco.cs.aalto.fi/publications/2005/makela-sema-nic-search-2005.pdf>
- Mangold C (2007) A survey and classification of semantic search approaches. Int J Metadata Semant Ontolog 2(1):23–34. <https://doi.org/10.1504/IJMSO.2007.015073>
- Manning CD (2011) Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In: Gelbukh AF (ed) Computational linguistics and intelligent text processing. Springer, Berlin/Heidelberg, pp 171–189
- Mika P (2015) On schema.org and why it matters for the web. IEEE Int Comput 19(4):52–55. <https://doi.org/10.1109/MIC.2015.81>
- Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. CoRR abs/1301.3781. <http://arxiv.org/abs/1301.3781>, 1301.3781
- Nalisnick E, Mitra B, Craswell N, Caruana R (2016) Improving document ranking with dual word embeddings. In: Proceedings of the 25th international conference companion on world wide web, International world wide web conferences steering committee, WWW'16 Companion. Republic and Canton of Geneva, Switzerland, pp 83–84. <https://doi.org/10.1145/2872518.2889361>
- Oren E, Delbru R, Catasta M, Cyganiak R, Stenzhorn H, Tummarello G (2008) Sindice.com: a document-oriented lookup index for open linked data. IJMSO 3(1):37–52. <https://doi.org/10.1504/IJMSO.2008.021204>
- Pehcevski J, Vercoustre AM, Thom JA (2008) Exploiting locality of wikipedia links in entity ranking. In: Macdonald C, Ounis I, Plachouras V, Ruthven I, White RW (eds) Advances in information retrieval. Springer, Berlin/Heidelberg, pp 258–269
- Pound J, Mika P, Zaragoza H (2010) Ad-hoc object retrieval in the web of data. In: Proceedings of the 19th international conference on world wide web, WWW'10. ACM, New York, pp 771–780. <http://doi.acm.org/10.1145/1772690.1772769>
- Prokofyev R, Tonon A, Luggen M, Vouilloz L, Difallah DE, Cudré-Mauroux P (2015) Sanaphor: ontology-based coreference resolution. In: Proceedings of the 14th international conference on the semantic web, ISWC 2015, vol 9366. Springer, Berlin/Heidelberg, pp 458–473. https://doi.org/10.1007/978-3-319-25007-6_27
- Rebele T, Suchanek FM, Hoffart J, Biega J, Kuzey E, Weikum G (2016) YAGO: a multilingual knowledge base from wikipedia, wordnet, and geonames. In: Proceedings Part II 15th international semantic web conference of the semantic web, ISWC 2016, Kobe, 17–21 Oct, pp 177–185. https://doi.org/10.1007/978-3-319-46547-0_19
- Ristoski P, Paulheim H (2016) Rdf2vec: Rdf graph embeddings for data mining. In: Groth P, Simperl E, Gray A, Sabou M, Krötzsch M, Lecue F, Flöck F, Gil Y (eds) The semantic web – ISWC 2016. Springer International Publishing, Cham, pp 498–514
- Rocha C, Schwabe D, Aragao MP (2004) A hybrid approach for searching in the semantic web. In: Proceedings of the 13th international conference on world wide web, WWW'04. ACM, New York, pp 374–383. <http://doi.acm.org/10.1145/988672.988723>

- Schuhmacher M, Ponzetto SP (2013) Exploiting dbpedia for web search results clustering. In: Proceedings of the 2013 workshop on automated knowledge base construction, AKBC'13. ACM, New York, pp 91–96. <http://doi.acm.org/10.1145/2509558.2509574>
- Schuhmacher M, Dietz L, Paolo Ponzetto S (2015) Ranking entities for web queries through text and knowledge. In: Proceedings of the 24th ACM international conference on information and knowledge management, CIKM'15. ACM, New York, pp 1461–1470. <http://doi.acm.org/10.1145/2806416.2806480>
- Sheth A, Bertram C, Avant D, Hammond B, Kochut K, Warke Y (2002) Managing semantic content for the web. IEEE Int Comput 6(4):80–87. <https://doi.org/10.1109/MIC.2002.1020330>
- Siencnik SK (2015) Adapting word2vec to named entity recognition. In: Proceedings of the 20th Nordic conference of computational linguistics, NODALIDA 2015, 11–13 May. Institute of the Lithuanian Language, Vilnius, pp 239–243. <http://aclweb.org/anthology/W/W15/W15-1830.pdf>
- Stojanovic N, Studer R, Stojanovic L (2003) An approach for the ranking of query results in the semantic web. In: Fensel D, Sycara K, Mylopoulos J (eds) The semantic web – ISWC 2003. Springer, Berlin/Heidelberg, pp 500–516
- Tonon A, Demartini G, Cudré-Mauroux P (2012) Combining inverted indices and structured search for ad-hoc object retrieval. In: Proceedings of the 35th international ACM SIGIR conference on research and development in information retrieval, SIGIR'12. ACM, New York, pp 125–134. <http://doi.acm.org/10.1145/2348283.2348304>
- Tran T, Cimiano P, Rudolph S, Studer R (2007) Ontology-based interpretation of keywords for semantic search. In: Proceedings of the 6th international the semantic web and 2nd asian conference on asian semantic web conference, ISWC'07/ASWC'07. Springer, Berlin/Heidelberg, pp 523–536. <http://dl.acm.org/citation.cfm?id=1785162.1785201>
- Voorhees EM (1993) Using wordnet to disambiguate word senses for text retrieval. In: Proceedings of the 16th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'93. ACM, New York, pp 171–180. <http://doi.acm.org/10.1145/160688.160715>
- Vrandecic D, Krötzsch M (2014) Wikidata: a free collaborative knowledgebase. Commun ACM 57(10):78–85. <http://doi.acm.org/10.1145/2629489>
- Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: a survey of approaches and applications. IEEE Trans Knowl Data Eng 29(12):2724–2743. <https://doi.org/10.1109/TKDE.2017.2754499>
- Zhang L, Yu Y, Zhou J, Lin C, Yang Y (2005) An enhanced model for searching in semantic portals. In: Proceedings of the 14th international conference on world wide web, WWW'05. ACM, New York, pp 453–462. <http://doi.acm.org/10.1145/1060745.1060812>

Semantic Stream Processing

Danh Le-Phuoc¹ and Manfred Hauswirth^{1,2}

¹Open Distributed Systems, Technical University of Berlin, Berlin, Germany

²Fraunhofer FOKUS, Berlin, Germany

Synonyms

RDF stream processing; Stream reasoning

Definitions

Semantic stream processing (SSP) refers to a set of models, principles, and techniques for analyzing and processing stream data by exploiting semantic structures which are explicitly or implicitly embedded in stream data elements. Such “semantic streams” are represented as sequences of temporal graphs linking human-machine understandable semantics and computational primitives. Semantic stream processing approaches leverage reasoning capabilities through formally defined rules to automate and optimize their continuous processing flows formulated in high-level abstract concepts and relationships.

Overview

Billions of sensors being distributed across the globe are continuously streaming data about the physical world around us. The stream data generated by networks of sensors enables us to detect and identify a multitude of things, from simple phenomena to complex events and situations. However, the lack of integration and communication between these networks and the lack of contextual information and background knowledge often isolate important data streams and intensify the existing problems of too much data and not enough knowledge about implicit meaning of such data and user intentions. As a potential remedy for such problems, Whitehouse et al. (2006) proposed the concept of “semantic streams” that

represent stream data associated with logic rules. Such logic rules represented in Prolog allow users to pose declarative queries over semantic interpretations of sensor data. Interestingly, the Prolog-based query model of Whitehouse et al. (2006) from the wireless sensor network community is adopted itself from the semantic Web services paradigm. This paradigm is similar to the concept of “semantic sensor Web” Sheth et al. (2008a) from the semantic Web community. Sheth et al. (2008a) claimed that sensors annotated with semantic metadata will increase interoperability and provide contextual information which is essential for situational knowledge.

As a result, sensor data published as semantic data sources (in RDF models) along with dynamic web data sources (social network, web blogs, etc.), which can be viewed as a variant of sensor data, introduced several research challenges in addressing their highly dynamic and low-latency processing nature. This then opened a new research trend in the semantic Web community, called RDF stream processing (RSP) or Linked Data stream processing (Le-Phuoc et al. 2012b), dealing with heterogeneous stream data sources that can be modeled by means of the RDF model. The choice of RDF as the data model, in combination with ontological profiles for representing stream data elements, offers not only interoperability but also well-understood semantics based on Datalog, Description Logics (DLs), and Answer Set Programming (ASP). Following this design choice, several RSP engines were built by extending existing semantic Web software stacks, e.g., triple storage and SPARQL query engines. Consequently, minimizing differences to “static” RDF and “static” SPARQL became the common goal for a majority of approaches in this line of work. Despite a significant amount of work dedicated to modeling stream data in RDF- and SPARQL-like continuous query languages in the last 10 years, none of them can provide a comprehensive, clean, and sound theoretical foundation along with a robust implementation.

However, amid the disagreement on a unified data model and query language, there are a lot of parallel efforts in building applications and processing engines in this domain. This movement

has started a new research trend of trying to realize reasoning features as offered in conventional RDF engines also for streams, called stream reasoning (Margara et al. 2014; Dell’Aglio et al. 2017). This research domain advocates stream reasoning as the common theme for enabling logic entailment profiles (RDFS, OWL, etc.) of RSP engines as well as other kinds of reasoning over RDF streams, e.g., statistical reasoning. There are few attempts toward this direction, however. The stream reasoning research area remains vastly unexplored, both from a theoretical point of view and also from the perspective of systems and tools supporting it. Notably, the implementations often are done before having a sound theoretical foundation.

In our context, a stream may consist of any kind of raw data with the restriction of discrete elements, e.g., we do not consider “normal” video or audio streams, but discrete pieces of such data. The same holds true for any other data which is discrete in nature, e.g., sensor readings. Semantic streams consist of such discrete elements which are semantically annotated. These annotations can be done manually or by way of automatic annotation through data analysis or hybrid approaches, e.g., the stream and its characteristics are described manually, whereas the individual stream elements are annotated by software. An example for this is monitoring of parking spaces with a camera, where the raw data is not interesting but only if a parking space is occupied or not. Thus the data is “lifted,” and this process is called semantic lifting which benefits greatly from the broad availability of machine learning tools, e.g., object recognition with convolution neural networks and entity extraction with natural language processing (NLP) tools. In fact, semantic lifting can tap into a wide range of semantic computing pipelines (Sheu et al. 2010) that has been around more than a decade. It is important to question whether it is trivial to seamlessly integrate these two types into a single processing pipeline. The constituting parts of stream reasoning are grounded in well-understood formal semantics and can usually be expressed via straightforward sets of rules. As such, they do not exhibit the complexity and the

opacity of artificial intelligence approaches that are based on machine learning and neural models.

In semantic stream processing, meaning and relationships do not have to be predefined and “hardwired” into data formats and the application program code at design time. Semantic technology enables encoding and extracting meaning separately from stream data elements and stream sources and separately from application code. With the interlinked nature of semantic data associated with stream data, a software agent can directly search topics, concepts, and associations that span a vast number of stream data sources linked with knowledge graphs. This automatic discovery capability fosters the dynamic composable in a semantic stream processing pipeline. Hence, adding, changing, and implementing new relationships or interconnecting sub-stream processing pipelines in a different way can be done on the fly at run-time after deploying the application logic. In this fashion, a semantic stream processing pipeline can be federated across autonomous processing agents which expose their self-describing stream sources and processing capabilities. Via automatic discovery, the software agent does not have to have full a priori knowledge about input data sources to deploy its application logic. For instance, an autonomous vehicle can continuously discover stream data sources available via its current network connections without knowing them or their types in advance. It can then subscribe the corresponding continuous queries to the relevant stream sources, e.g., to get notifications about traffic jams on the roads and junctions nearby its locations without having to hard-code the data sources and queries.

Key Research Findings

The most consolidated group of research findings on semantic stream processing includes contributions from RSP engines and stream reasoners. These contributions aim at enabling the semantic integration of heterogeneous stream data sources with (large) static datasets stored

in relational databases or triple stores through declarative continuous queries based on RDF and SPARQL. By adding window operators to SPARQL and extending the RDF model to capture temporal aspects of stream elements, RSP engines such as C-SPARQL (Barbieri et al. 2010b), CQELS (Le-Phuoc et al. 2011) and SPARQL_{stream} (Calbimonte et al. 2010) integrate the features of data stream management systems (DSMSs) and SPARQL engines. For their implementation, different approaches were chosen based on the features or performance criteria of interest. In the top-down approach, query features are broken down for delegation to the underlying processing engines. The extreme case following the top-down approach is rewriting a continuous query in SPARQL form to SQL queries on traditional relational database or relational data stream management systems. This approach is called ontology-based data access (ODBA), and ODBA-based engines (Calbimonte et al. 2010; Kharlamov et al. 2017) use the ontologies as the semantic guidelines for rewriting their queries to a targeted database schema. EP-SPARQL (Anicic et al. 2010) is another variant of this rewriting approach which translates its unified language for event processing and reasoning as logic expressions to Prolog rules. These are then executed in a Prolog engine. In contrast to this, the bottom-up approach builds or reuses physical query operators to construct execution engines which coordinate and control the query execution process. It takes considerably more effort to build query engines of this kind, e.g., CQELS and C-SPARQL. However, it is easier to improve the performance and add more features as the implementation is under full control. For example, C-SPARQL extended reasoning features, and CQELS implemented more efficient data structures to improve processing throughput.

To offer a high processing throughput, the system implementers need to spend significant engineering efforts on optimizing and tuning physical data structures (Le-Phuoc 2017; Ren et al. 2017) and query executors (Le-Phuoc et al. 2013; Ren et al. 2017; Bazoobandi et al. 2017) tailored to the

RDF graph nature of stream elements. To scale out the processing to multiple processing nodes, generic cloud-based stream platforms are used: For example, CQELS Cloud (Le-Phuoc et al. 2013) uses Apache Storm and Strider (Ren et al. 2017) uses Apache Spark. The scaling strategy of Strider offers a good solution for handling computationally expensive operations like continuous reasoning on RDFS+ profiles.

In the continuous query setting, the incremental reasoning approach is the obvious choice for materializing the logic entailments of the RDF data within a window. With the observation that deletions can be foreseen and are not random, expiration time annotations are associated with all the axioms involved in the materialization, and such information is exploited to identify only the facts to be deleted. Sparkwave (Komazec et al. 2012) makes use of the RETE (Forgy 1982) algorithm to maintain RDFS entailments with sliding windows defined in C-SPARQL whereby RDFS axioms are encoded as RETE rules and organized in a network. A different approach is adopted by INSTANS (Rinne et al. 2016) which adopts SPARQL 1.1 with an extension to its query evaluation model to continuously query data streams. The implementation of INSTANS also relies on the RETE algorithm: Tasks are expressed as networks of queries and compiled in RETE-like structures to evaluate the results. When new facts are added to the system, they are matched against these rules.

When stream reasoning started to take off in the scientific community, various efforts tried to unify the query language primitives to promote a query model with a sound formalization which can generalize existing query languages such as C-SPARQL, CQELS-QL, and EP-SPARQL. For example, Dell’Aglio et al. (2014) targeted unifying query languages for RSP engines and extended the support for CEP (Dell’Aglio et al. 2016). In parallel, Beck et al. (2015) proposed the LARS framework as a unified way to express stream reasoning primitives under ASP foundations. LARS was later implemented in Laser (Bazoobandi et al. 2017). From a Datalog perspective, windowing operators are just a special case of temporal Datalog variants, so Ronca et al.

(2018) recently proposed a Datalog-based logic framework for stream reasoning with a comprehensive complexity analysis.

An additional and noteworthy extension to logic-based reasoning is presented in Barbieri et al. (2010a) where the authors focus on inductive stream reasoning. Inductive stream reasoning involves mining of large portions of data and applying statistical and machine learning techniques to extract new knowledge. The authors propose combining inductive reasoning with deductive reasoning to increase accuracy of the inductive reasoning. The technique has been applied and validated on a real scenario derived from social media analysis, showing the accuracy of the reasoning algorithm in this context. In a similar effort, Chen et al. (2017) showed that DL-based reasoning can integrate efficiently with online streaming mining for traffic data.

Benefiting from the semantic Web adoption, Le-Phuoc et al. (2012a) and Arias Fisteus et al. (2014) proposed middleware solutions transforming and enriching unstructured data streams or raw sensory data to RDF streams by reusing existing tools from the semantic Web stack. Together with other trends on exposing IoT streams using ontologies (Barnaghi et al. 2012), the research community is working very actively on defining suitable ontologies for capturing semantics of sensor data and its contextual information. Examples are the W3C Semantic Sensor Network Ontology (Haller et al. 2017) and the Thing Description (Kaebisc and Kamiya 2018) and Time Ontology (Cox and Little 2017). This lays a solid foundation for semantic lifting which can use available tools for detecting semantic events (Rea et al. 2004; Wang et al. 2006; Chang et al. 2015) and extracting semantic relationships, e.g., from video scenes (Baier et al. 2017), to generate semantic streams.

Examples of Application

Internet of Things (IoT) and Smart Cities: IoT and smart cities have an extremely wide application range where the ideas of semantic sen-

sor Web (Sheth et al. 2008b) and semantic sensor stream (Barnaghi et al. 2013) can be applicable for various applications. In smart cities, most of the applications try to process and understand information relevant for the life of people in a city and use it to make the city more efficient, friendly to the environment, etc. Such applications have access to a huge amount of heterogeneous stream sensor sources. For example, traffic events from social streams (Anantharam et al. 2015) and event streams of big cities (Anantharam et al. 2016), traffic streams (Chen et al. 2017; Eiter et al. 2017), and other IoT streams (Puschmann et al. 2017) are extremely dynamic data sources which demand efficient and scalable algorithms for processing data in near-real-time at the semantic level. In this context, the semantic stream processing must provide enough expressiveness to abstract and derive high-level concepts from low-level and time-annotated data. Moreover, smart cities require the large-scale integration of different data types and sources: As an example, traffic information can be retrieved from sensors, through tracking cell phones in GSM cells, as well as from navigation systems, or posts on social networks.

Another interesting concept is **citizen sensing** which is social sensing enabled by mobile sensors and human computing whereby humans and their devices act as “sensors” and share their observations and views using mobile devices and Web 2.0 services (Sheth 2009). Together with live social semantics (Alani et al. 2009) and social media stream (Balduini et al. 2012), the applications of this type requires semantic analysis of social media by extending traditional analysis based on graphs enriching the connections between people and concepts with semantic annotations. One of the goals of the analysis of social media is to capture hidden relations between people and concepts. Some experiments on the use of social media analysis have been reported in Balduini et al. (2013), where the authors focus on the processing of Twitter posts to extract new information, e.g., how the mood changes during the day, which are the trending topics, etc., during large-scale events (London Olympic Games 2012 and Milano Design Week 2013).

Predictive Maintenance: Sensing devices allow industrial applications to actively and constantly monitor machines to predict maintenance cycles. Such applications include a large number of sensor sources with a wide variety of sensing platforms and types of measurements, not to mention large enterprise knowledge bases, that need to be correlated with stream data to enrich such time series data. For instance, in Kharlamov et al. (2016) the data processing task requires to extract, aggregate, and correlate static data about turbine structure, streaming data produced by up to 2,000 sensors installed in different parts of the turbine, and historical operational data of the reference sensors stored in multiple data sources. Similarly, the application scenarios from Siemens Energy (Kharlamov et al. 2017) and the DEBS challenge 2017 (Gulisano et al. 2017) need efficient and scalable semantic stream processing pipelines to detect abnormal behavior in manufacturing machines and provide actionable insights based on the observations in time.

Future Directions for Research

As an increasing number of semantic streams is made available for a new generation of applications, practitioners are building more and more engines supporting SSP or integrate SSP into current stream engines or RDF stores. To comprehensively support all features of current RSP engines, CEP, and stream reasoning engines, a top-down approach would provide the theoretical foundations associated with feasible implementations for a new generation of SSP engines in the years to come. As shown in the surveys of Margara et al. (2014) and Dell’Aglio et al. (2017), a unified data model associated with well-formulated semantic processing primitives is a vital prerequisite for building a generic execution framework for SSP. The analysis of computational complexity of the resulting processing models should be done at a fine-grain level to give effective guidelines for designing and implementing the underlying engines.

The first challenging problem in generalizing the data model for stream elements or atomic events is capturing their order in the most generic way. As shown in Margara et al. (2014) and Dell’Aglio et al. (2017), current time models based on logical timestamps have their own shortcomings in terms of expressivity and practical implementations. A solution for enforcing causal order in both windowing operators and sequence patterns of events has to be seamlessly integrated with soft-orders entailed by semantic rules. This solution has to take into account out-of-order streams/events, distributed settings, and imprecise timestamps and imprecise clocks. Then the uncertainty properties need to be captured along with semantic relationships and concepts, especially, since there are more and more symbolic values/entities generated by machine learning algorithms from discrete noisy data as semantic streams. Uncertainty can be modeled together with other provenance information which can be represented as ontological contextual information.

Together with accommodating traditional computing primitives from CEP, SPARQL, DSMS, etc., interleaving reasoning primitives without violating the decidability of the whole processing pipeline is a challenging research problem in designing a declarative query language or domain-specific languages (DSLs) for SSP. Some proposals (Barbieri et al. 2010a; Chen et al. 2017) have started investigating the use of inductive and/or statistical reasoning. Also, recent developments in natural language understanding and computer vision have paved the way for a new generation of semantic query capabilities over stream data, e.g., social streams and video streams. Specifically, the visual reasoning capabilities as proposed in Johnson et al. (2017) and Jang et al. (2017) enable the integration of visual question answering (VQA) query fragments into a SSP pipeline. Interestingly, there are various potential links among VQA and NLP and semantic Web. For instance, to improve some state-of-the-art techniques of VQA, Baier et al. (2017) model scene descriptions through RDF triples

which are used to answer the queries about visual relationships in human language form as specified in the Visual Genome dataset (Krishna et al. 2016). It is worth noting that the data and queries of the Visual Genome can be naturally expressed in RDF graphs and SPARQL queries, respectively. On the other hand, there are plenty of works on question answering in human language form for RDF or knowledge graphs, e.g., Unger et al. (2012) and Bordes et al. (2014). The aforementioned features are quite appealing for developers amid the rise of transfer learning with a plethora of free pretrained models, but the computational complexity implications need to be investigated thoroughly as well, as they are not yet well understood.

Increasing the expressiveness of how to define a query or a processing pipeline on semantic streams will incur further research and engineering challenges as achieving good performance, and scalability will be necessary and the progress in this domain is rather modest at the moment. For improving performance, the research problems are not limited to designing more efficient algorithms for certain types of computation primitives, e.g., incremental reasoning in windowing data collections, but also include the execution settings such as processing load, resource constraints, and parallelization and coordination assumptions. For instance, continuous optimization considering several dynamic processing aspects such as multiple queries, volume of static data, and input rates is rather challenging even with traditional relational data stream processing. On top of that, a common assumption of processing stream data is that the processing state can be completely loaded into main memory. This might be overly optimistic, and standard techniques like storing to disks or shifting the data and processing to other computing nodes should be considered as optimization strategies. This new working assumption is advocated by the edge computing paradigm which matches the distributed nature of a majority of stream applications. Also and in particular, the federated processing model of RSP will bring new challenges for current information systems, such as the Web. Uniform parallel

computing settings like cloud/private clusters and high-performance computing nodes with large numbers of CPUs and GPUs will play an important role for continuously handling intensive workloads triggered by stream data at Web scale. This provides more options and capabilities along with technical and research challenges in realizing them.

Cross-References

- ▶ [Adaptive Windowing](#)
- ▶ [Continuous Queries](#)
- ▶ [Definition of Data Streams](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Stream Processing Languages and Abstractions](#)
- ▶ [Linked Data Management](#)
- ▶ [Linked Geospatial Data](#)
- ▶ [Semantic Interlinking](#)
- ▶ [Stream Query Optimization](#)
- ▶ [Stream Window Aggregation Semantics and Optimization](#)
- ▶ [Streaming Big Spatial Data](#)
- ▶ [Types of Stream Processing Algorithms](#)

References

- Alani H, Szomszor M, Cattuto C, Van den Broeck W, Correndo G, Barrat A (2009) Live social semantics. Springer, Berlin/Heidelberg, pp 698–714. https://doi.org/10.1007/978-3-642-04930-9_44
- Anantharam P, Barnaghi P, Thirunarayan K, Sheth A (2015) Extracting city traffic events from social streams. ACM Trans Intell Syst Technol 6(4):43:1–43:27. <https://doi.org/10.1145/2717317>
- Anantharam P, Thirunarayan K, Marupudi S, Sheth A, Banerjee T (2016) Understanding city traffic dynamics utilizing sensor and textual observations. In: Proceedings of the thirtieth AAAI conference on artificial intelligence (AAAI'16). AAAI Press, pp 3793–3799. <http://dl.acm.org/citation.cfm?id=3016387.3016438>
- Anicic D, Fodor P, Rudolph S, Stühmer R, Stojanovic N, Studer R (2010) A rule-based language for complex event processing and reasoning. In: Proceedings of the fourth international conference on web reasoning and rule systems (RR'10). Springer, Berlin/Heidelberg, pp 42–57
- Arias Fisteus J, Fernández García N, Sánchez Fernández L, Fuentes-Lorenzo D (2014) Zstreamy. Web Semant 25(C):16–23. <https://doi.org/10.1016/j.websem.2013.11.002>
- Baier S, Ma Y, Tresp V (2017) Improving visual relationship detection using semantic modeling of scene descriptions. In: d'Amato C, Fernández M, Tamia VAM, Lécué F, Cudré-Mauroux P, Sequeda JF, Lange C, Heflin J (eds) The semantic web – ISWC 2017 – proceedings of 16th international semantic web conference, Vienna, 21–25 Oct 2017, Part I. Lecture notes in computer science, vol 10587, pp 53–68. https://doi.org/10.1007/978-3-319-68288-4_4
- Baldoni M, Celino I, Dell'Aglio D, Della Valle E, Huang Y, Lee T, Kim SH, Tresp V (2012) Bottari: an augmented reality mobile application to deliver personalized and location-based recommendations by continuous analysis of social media streams. Web Semant 16:33–41. <https://doi.org/10.1016/j.websem.2012.06.004>
- Baldoni M, Della Valle E, Dell'Aglio D, Tsytarau M, Palpanas T, Confalonieri C (2013) Social listening of City scale events using the streaming linked data framework. Springer, Berlin/Heidelberg, pp 1–16. https://doi.org/10.1007/978-3-642-41338-4_1
- Barbieri D, Braga D, Ceri S, Valle ED, Huang Y, Tresp V, Rettinger A, Wermser H (2010a) Deductive and inductive stream reasoning for semantic social media analytics. IEEE Intell Syst 25(6):32–41. <https://doi.org/10.1109/MIS.2010.142>
- Barbieri DF, Braga D, Ceri S, Grossniklaus M (2010b) An execution environment for C-SPARQL queries. In: EDBT 2010, pp 441–452
- Barnaghi P, Wang W, Henson C, Taylor K (2012) Semantics for the internet of things: early progress and back to the future. Int J Semant Web Inf Syst 8(1):1–21. <https://doi.org/10.4018/jswis.2012010101>
- Barnaghi P, Wang W, Dong L, Wang C (2013) A linked-data model for semantic sensor streams. In: 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE Cyber, physical and social computing, pp 468–475. <https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.95>
- Bazoobandi HR, Beck H, Urbani J (2017) Expressive stream reasoning with laser. CoRR abs/1707.08876. <http://arxiv.org/abs/1707.08876>
- Beck H, Dao-Tran M, Eiter T, Fink M (2015) Lars: a logic-based framework for analyzing reasoning over streams. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence (AAAI'15). AAAI Press, pp 1431–1438. <http://dl.acm.org/citation.cfm?id=2887007.2887205>
- Bordes A, Chopra S, Weston J (2014) Question answering with subgraph embeddings. CoRR abs/1406.3676. <http://arxiv.org/abs/1406.3676>
- Calbimonte JP, Corcho O, Gray AJG (2010) Enabling ontology-based access to streaming data sources. In: Proceedings of the 9th international seman-

- tic web conference on the semantic web – volume part I (ISWC'10). Springer, Berlin/Heidelberg, pp 96–111
- Chang X, Yang Y, Xing EP, Yu YL (2015) Complex event detection using semantic saliency and nearly-isotonic SVM. In: Proceedings of the 32nd international conference on international conference on machine learning, vol 37, JMLR.org (ICML'15), pp 1348–1357. <http://dl.acm.org/citation.cfm?id=3045118.3045262>
- Chen J, Lécué F, Pan JZ, Chen H (2017) Learning from ontology streams with semantic concept drift. In: Sierra C (ed) Proceedings of the twenty-sixth international joint conference on artificial intelligence (IJCAI 2017), Melbourne, 19–25 Aug 2017, ijcai.org, pp 957–963. <https://doi.org/10.24963/ijcai.2017/133>
- Cox S, Little C (2017) Time ontology in owl. <https://www.w3.org/TR/owl-time/>. Online; Accessed 21 Mar 2018
- Dell'Aglio D, Valle ED, Calbimonte J, Corcho Ó (2014) RSP-QL semantics: a unifying query model to explain heterogeneity of RDF stream processing systems. Int J Semant Web Inf Syst 10(4):17–44. <https://doi.org/10.4018/ijswis.2014100102>
- Dell'Aglio D, Dao-Tran M, Calbimonte JP, Le-Phuoc D, Della Valle E (2016) A query model to capture event pattern matching in RDF stream processing query languages. Springer, Cham, pp 145–162. https://doi.org/10.1007/978-3-319-49004-5_10
- Dell'Aglio D, Della Valle E, van Harmelen F, Bernstein A (2017) Stream reasoning: a survey and outlook. Data Sci 01(1–2):59–83
- Eiter T, Parreira JX, Schneider P (2017) Spatial ontology-mediated query answering over mobility streams. Springer, Cham, pp 219–237. https://doi.org/10.1007/978-3-319-58068-5_14
- Forgy CL (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell 19(1):17–37. [https://doi.org/10.1016/0004-3702\(82\)90020-0](https://doi.org/10.1016/0004-3702(82)90020-0). <http://www.sciencedirect.com/science/article/pii/0004370282900200>
- Gulisano V, Jerzak Z, Katerinenko R, Strohbach M, Ziekow H (2017) The DEBS 2017 grand challenge. In: Proceedings of the 11th ACM international conference on distributed and event-based systems (DEBS'17). ACM, New York, pp 271–273. <https://doi.org/10.1145/3093742.3096342>
- Haller A, Janowicz K, Cox S, Phuoc DL, Taylor K, Lefrançoi M (2017) Semantic sensor network ontology, w3c recomendation. <https://www.w3.org/TR/vocab-ssn/>. Online; Accessed 21 Mar 2018
- Jang Y, Song Y, Yu Y, Kim Y, Kim G (2017) TGIF-QA: toward spatio-temporal reasoning in visual question answering. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR 2017), Honolulu, 21–26 July 2017, pp 1359–1367. <https://doi.org/10.1109/CVPR.2017.149>
- Johnson J, Hariharan B, van der Maaten L, Hoffman J, Fei-Fei L, Zitnick CL, Girshick RB (2017) Inferring and executing programs for visual reasoning. In: IEEE international conference on computer vision (ICCV 2017), Venice, 22–29 Oct 2017. IEEE Computer Society, pp 3008–3017. <https://doi.org/10.1109/ICCV.2017.325>
- Kaebisc S, Kamiya T (2018) Web of things (wot) thing description. <https://www.w3.org/TR/wot-thing-description/>. Online; Accessed 21 Mar 2018
- Kharlamov E, Kotidis Y, Mailis T, Neuenstadt C, Nikolaou C, Özcep Ö, Svingos C, Zheleznyakov D, Brandt S, Horrocks I, Ioannidis Y, Lamparter S, Möller R (2016) Towards analytics aware ontology based access to static and streaming data. Springer, Cham, pp 344–362. https://doi.org/10.1007/978-3-319-46547-0_31
- Kharlamov E, Mailis T, Mehdi G, Neuenstadt C, Özcep Ö, Roshchin M, Solomakhina N, Soylu A, Svingos C, Brandt S, Giese M, Ioannidis Y, Lamparter S, Möller R, Kotidis Y, Waaler A (2017) Semantic access to streaming and static data at siemens. Web Semant Sci Serv Agents World Wide Web 44(Suppl C):54–74. <https://doi.org/10.1016/j.websem.2017.02.001>. <http://www.sciencedirect.com/science/article/pii/S1570826817300124>; Industry and In-use Applications of Semantic Technologies
- Komazec S, Cerri D, Fensel D (2012) Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In: Proceedings of the 6th ACM international conference on distributed event-based systems (DEBS'12). ACM, New York, pp 58–68. <https://doi.org/10.1145/2335484.2335491>
- Krishna R, Zhu Y, Groth O, Johnson J, Hata K, Kravitz J, Chen S, Kalantidis Y, Li LJ, Shamma DA, Bernstein M, Fei-Fei L (2016) Visual genome: connecting language and vision using crowdsourced dense image annotations. <https://arxiv.org/abs/1602.07332>
- Le-Phuoc D (2017) Operator-aware approach for boosting performance in RDF stream processing. Web Semant Sci Serv Agents World Wide Web 42(Suppl C):38–54. <https://doi.org/10.1016/j.websem.2016.04.001>. <http://www.sciencedirect.com/science/article/pii/S1570826816300014>
- Le-Phuoc D, Dao-Tran M, Parreira JX, Hauswirth M (2011) A native and adaptive approach for unified processing of linked streams and linked data. In: Proceedings of 10th international semantic web conference, pp 370–388
- Le-Phuoc D, Nguyen-Mau HQ, Parreira JX, Hauswirth M (2012a) A middleware framework for scalable management of linked streams. Web Semant Sci Serv Agents World Wide Web 16(Suppl C):42–51. <https://doi.org/10.1016/j.websem.2012.06.003>. <http://www.sciencedirect.com/science/article/pii/S1570826812000728>; the Semantic Web Challenge 2011
- Le-Phuoc D, Xavier Parreira J, Hauswirth M (2012b) Linked stream data processing. Springer, Berlin/Heidelberg, pp 245–289. https://doi.org/10.1007/978-3-642-33158-9_7
- Le-Phuoc D, Quoc HNM, Van CL, Hauswirth M (2013) Elastic and scalable processing of linked stream data in the cloud. In: ISWC 2013 (1), pp 280–297. https://doi.org/10.1007/978-3-642-41335-3_18

- Margara A, Urbani J, van Harmelen F, Bal H (2014) Streaming the web: reasoning over dynamic data. *Web Semant Sci Serv Agents World Wide Web* 25(Suppl C):24–44. <https://doi.org/10.1016/j.websem.2014.02.001>. <http://www.sciencedirect.com/science/article/pii/S1570826814000067>
- Puschmann D, Barnaghi P, Tafazolli R (2017) Adaptive clustering for dynamic IoT data streams. *IEEE Internet Things J* 4(1):64–74. <https://doi.org/10.1109/JIOT.2016.2618909>
- Rea N, Dahyot R, Kokaram A (2004) Semantic event detection in sports through motion understanding. Springer, Berlin/Heidelberg, pp 88–97. https://doi.org/10.1007/978-3-540-27814-6_14
- Ren X, Curé O, Ke L, Lhez J, Belabbess B, Randriamalala T, Zheng Y, Kepkelian G (2017) Strider: an adaptive, inference-enabled distributed RDF stream processing engine. *Proc VLDB Endow* 10(12):1905–1908. <https://doi.org/10.14778/3137765.3137805>
- Rinne M, Solanki M, Nuutila E (2016) Rfid-based logistics monitoring with semantics-driven event processing. In: Proceedings of the 10th ACM international conference on distributed and event-based systems (DEBS’16). ACM, New York, pp 238–245. <https://doi.org/10.1145/2933267.2933300>
- Ronca A, Kaminski M, Cuenca Grau B, Motik B, Horrocks I. Stream reasoning in temporal datalog. In: Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI 2018). AAAI Press, New Orleans
- Sheth A (2009) Citizen sensing, social signals, and enriching human experience. *IEEE Internet Comput* 13(4):87–92. <https://doi.org/10.1109/MIC.2009.77>
- Sheth A, Henson C, Sahoo SS (2008a) Semantic sensor web. *IEEE Internet Comput* 12(4):78–83. <https://doi.org/10.1109/MIC.2008.87>
- Sheth AP, Henson CA, Sahoo SS (2008b) Semantic sensor web. *IEEE Internet Comput* 12(4):78–83
- Sheu P, Yu H, Ramamoorthy CV, Joshi AK, Zadeh LA (2010) Semantic computing. Wiley-IEEE Press, New York
- Unger C, Bühlmann L, Lehmann J, Ngomo AC, Gerber D, Cimiano P (2012) Template-based question answering over RDF data. In: Proceedings of the 21st international conference on world wide web (WWW’12). ACM, New York, pp 639–648. <https://doi.org/10.1145/2187836.2187923>
- Wang T, Li J, Diao Q, Hu W, Zhang Y, Dulong C (2006) Semantic event detection using conditional random fields. In: Proceedings of the 2006 conference on computer vision and pattern recognition workshop (CVPRW’06), IEEE Computer Society, Washington, DC, pp 109–114. <https://doi.org/10.1109/CVPRW.2006.190>
- Whitehouse K, Zhao F, Liu J (2006) Semantic streams: a framework for composable semantic interpretation of sensor data. In: Proceedings of the third European conference on wireless sensor networks (EWSN’06). Springer, Berlin/Heidelberg, pp 5–20. https://doi.org/10.1007/11669463_4

Sentiment-Based Privacy Preserving Data Publishing Techniques for Social Networks

► Privacy Cube

Similarity Estimation

► Similarity Sketching

Similarity Sketching

Rasmus Pagh

Computer Science Department, IT University of Copenhagen, Copenhagen S, Denmark

Synonyms

Distance estimation; Similarity estimation; Similarity summarization

Overview

Similarity between a pair of objects, usually expressed as a *similarity score* in [0, 1], is a key concept when dealing with noisy or uncertain data, as is common in big data applications.

The aim of *similarity sketching* is to estimate similarities in a (high-dimensional) space using fewer computational resources (time and/or storage) than a naïve approach that stores unprocessed objects. This is achieved using a form of lossy compression that produces succinct representations of objects in the space, from which similarities can be estimated. In some spaces, it is more natural to consider *distances* rather than similarities; we will consider both of these measures of proximity in the following.

Definitions

Formally, consider a space X of objects and a function $d : X \times X \rightarrow \mathbf{R}_+$. We refer to d as a *distance function* for X . Similarity sketching with respect to (X, d) is done by using a sketching function $c : X \rightarrow \{0, 1\}^s$ such that for every pair of objects $x_1, x_2 \in X$, the distance $d(x_1, x_2)$ can be approximated from the knowledge of $c(x_1)$ and $c(x_2)$. Most forms of similarity sketching are *randomized* and produce an estimate $\hat{d}(c(x_1), c(x_2))$ such that with probability $1 - \delta$, it holds that:

$$\begin{aligned} (1 - \varepsilon) d(x_1, x_2) &\leq \hat{d}(c(x_1), c(x_2)) \\ &\leq (1 + \varepsilon) d(x_1, x_2), \quad (1) \end{aligned}$$

where $\varepsilon, \delta > 0$ are user-specified parameters. Often $c(x)$ itself represents an object in X , intuitively one that is close to x , and \hat{d} is just distance function evaluation. In some cases, it is not possible to achieve the kind of *multiplicative* error guarantee as in (1), and it is instead the case that $\hat{d}(c(x_1), c(x_2))$ differs from $d(x_1, x_2)$ by an additive constant in $[-\varepsilon, \varepsilon]$.

If X is finite, we note that the probability that some distance in X is *not* within the interval (1) is at most $\binom{|X|}{2}\delta$. If this probability is strictly less than 1, it is possible to fix a choice of sketching function c such that all distances estimated fulfill (1). In the case where X is a data set of interest, *data-dependent* similarity sketching methods can improve performance when estimating distances in X ; we do not describe such methods here – for more information, see Wang et al. (2017).

Key Techniques

Quantization

In signal processing, an object x from a large or infinite space X can be mapped to a nearby object $c(x)$ in a smaller, finite subset of the space, to enable a succinct representation. Such a mapping, referred to as *quantization*, can be used for similarity sketching by estimating $d(x, y)$ simply as

$d(c(x), c(y))$. An efficient quantization method for Euclidean and angular distances is *product quantization* by Jégou et al. (2011).

MinHash

An early form of similarity sketching targeted *Jaccard similarity* of sets. The technique, now known as *MinHash*, was introduced in Broder et al. (1997) and Broder (1997). It works as follows: For a random permutation $r : X \rightarrow X$, map a set S to the “MinHash value” $h_r(S) = \min_{x \in S} r(x)$, where the minimum is over an arbitrary, fixed ordering of X . In practice it is hard to construct random permutations, so instead one uses a hash function $r : X \rightarrow R$ where R is a large range (e.g., the 64-bit integers) ensuring that collisions in S are unlikely.

It can be shown that if sets S_1 and S_2 have Jaccard similarity J , the probability that $h_r(S_1) = h_r(S_2)$ (i.e., that the MinHash values collide) is J . MinHash is a so-called locality-sensitive hash function (LSH) for which collision probability depends on similarity (or distance). Computing h_r independently k times, with different hash functions r , we expect that the number t of collisions is close to kJ with high probability. Thus, we can estimate J well as $\hat{J} = t/k$, with precision that grows with k .

Improvements. MinHash can be improved in several ways. First of all, the space usage can be reduced by a method called *b-bit MinHash*, which consists of storing a b -bit hash signature rather than the MinHash value itself, yielding collision probability $J + (1 - J)/2^b$. The larger collision probability can be taken into account to derive an unbiased estimator for J . Ultimately, b -bit MinHash has greater precision at the same space usage compared to MinHash; see Li and König (2011).

The value of b used in practice is often 1 or 2, with space typical usage of $k = 32$ or $k = 64$ bits. Larger values of b are relevant when estimating Jaccard similarities close to 0. For $b = 1$, it is particularly efficient to compute the number of hash collisions, using bitwise exclusive or in conjunction with a `popcnt` that computes the

number of nonzero bits. When the Jaccard similarity of interest is above 0.8, there are methods that are more precise than b -bit MinHash, for example, *Odd Sketch* due to Mitzenmacher et al. (2014).

The time required for computing k MinHash values (or b -bit MinHash values) can be reduced by the so-called *one-permutation* method that works by initially splitting the set S into k parts and then computing a MinHash value for each part as described by Li et al. (2012). Another approach is *bottom-k sampling*, where the k smallest hash values of a single hash function are used to select a sample; see Thorup (2013) and its references. These methods have been further improved by Dahlgaard et al. (2017).

LSHable Distance Measures

Jaccard similarity is an example of an *LSHable* similarity measure, namely, it allows a random hash function r such that the collision probability equals the similarity. In fact, the MinHash method extends to any LSHable similarity measure by simply replacing the hash function r . An important LSHable distance measure is *angular distance* as shown by Charikar (2002). Angular distance, in turn, can be used to estimate *coseine similarity*. For more discussion of LSHable similarity measures, see Chierichetti and Kumar (2015).

For distance measures that allow an LSH, a similar result can be achieved since the probability of an LSH collision $r(x) = r(y)$ is a decreasing function of $d(x, y)$. The highest accuracy is achieved around distances for which the collision probability is not far from 1/2, say, in the range [1/3, 2/3]. Functions of this form, for example, for Hamming distance or Euclidean distance, are usually found in the literature on search data structures using locality-sensitive hashing; see, e.g., Andoni and Indyk (2008) and its references.

We observe that these similarity sketches can be efficiently computed in a *streaming* setting where only a single element of S is considered at a time. More information on approximate computing for stream analytics can be found elsewhere in this volume.

Dimension Reduction and Embeddings

A special case of similarity sketching is *dimension reduction*: mapping objects to a lower-dimensional space, reducing the representation size, while approximately preserving distances. We refer to the article on dimension reduction in this encyclopedia for more details.

We mention a particular form of dimension reduction called *kernel approximations* where distances after the embedding reflect a distance measure (called a *kernel*) on the original space. Such mappings are possible for a wide class of distance measures defined on Euclidean space; see, e.g., Rahimi and Recht (2007).

Some spaces X' can be *embedded* into a space X for which similarity sketching is possible, generally introducing a distortion of distances. For example, Manhattan (or ℓ_1) distances on $[0, 1]^d$ can be embedded into Hamming space by unary encoding of (quantized) coordinate values; see, e.g., Gionis et al. (1999). An embedding implies a similarity sketch on X' obtained by combining the embedding with a similarity sketch for X . Of course, the precision that can be obtained in this way is limited by the precision of the embedding.

Applications

Similarity sketching is used, for example:

- in situations where exact distance computation is not possible for reasons of space (e.g., main memory indexes supporting near neighbor search, where the set of vectors is too large to fit in the memory),
- when exact computation is undesirable for reasons of time (e.g., when considering a large set of candidate very high-dimensional objects in an algorithm for nearest neighbor search),
- as a preprocessing step in machine learning applications where it serves to reduce dimensionality as well as data size.

Since similarity sketching distorts distances, it will generally change the output compared to an exact algorithm and might introduce false positives as well as false negatives in search applications.

Software Libraries

Similarity sketching is often implemented as part of a particular application, and as such there does not seem to be many general-purpose libraries available. For Euclidean distances among a fixed set of points, the QuadSketch C++ library (<https://github.com/talwagner/quadsketch>) provides state-of-the-art performance with a data-dependent mapping. The DataSketch library (<https://github.com/ekzhu/dataskeetch>) provides Python implementations of variants of MinHash.

Cross-References

- ▶ Approximate Computing for Stream Analytics
- ▶ Dimension Reduction
- ▶ Query Processing – k NN
- ▶ Record Linkage

Acknowledgements This work received support from the European Research Council under the European Union’s 7th Framework Programme (FP7/2007-2013)/ERC grant agreement no. 614331.

References

- Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun ACM* 51(1):117–122
- Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of compression and complexity of sequences. IEEE, pp 21–29
- Broder AZ, Glassman SC, Manasse MS, Zweig G (1997) Syntactic clustering of the web. *Comput Netw ISDN Syst* 29(8):1157–1166
- Charikar M (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of symposium on theory of computing (STOC), pp 380–388
- Chierichetti F, Kumar R (2015) Lsh-preserving functions and their applications. *J ACM* 62(5):33
- Dahlgaard S, Knudsen MBT, Thorup M (2017) Fast similarity sketching. In: Proceedings of symposium on foundations of computer science (FOCS), pp 663–671
- Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of conference on very large databases (VLDB), pp 518–529
- Jégou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. *IEEE Trans Pattern Anal Mach Intell* 33(1):117–128

- Li P, König AC (2011) Theory and applications of b-bit minwise hashing. *Commun ACM* 54(8):101–109
- Li P, Owen AB, Zhang C (2012) One permutation hashing. In: Advances in neural information processing systems (NIPS), pp 3122–3130
- Mitzenmacher M, Pagh R, Pham N (2014) Efficient estimation for high similarities using odd sketches. In: Proceedings of international world wide web conference (WWW), pp 109–118
- Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Advances in neural information processing systems (NIPS), pp 1177–1184
- Thorup M (2013) Bottom- k and priority sampling, set similarity and subset sums with minimal independence. In: Proceedings of symposium on theory of computing (STOC). ACM, pp 371–380
- Wang J, Zhang T, Song J, Sebe N, Shen HT (2017) A survey on learning to hash. *IEEE Trans Pattern Anal Mach Intell* 13(9) <https://doi.org/10.1109/TPAMI.2017.2699960>

Similarity Summarization

- ▶ Similarity Sketching

Sliding-Window Aggregation Algorithms

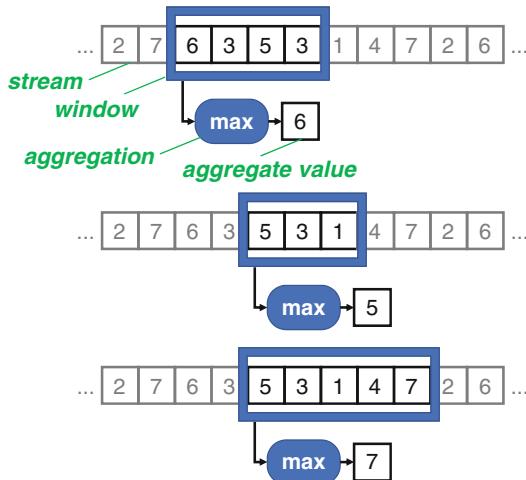
- Kanat Tangwongsan¹, Martin Hirzel², and Scott Schneider²
- ¹Mahidol University International College, Salaya, Thailand
- ²IBM Research AI, Yorktown Heights, NY, USA

Synonyms

- Sliding-window fold; Stream reduce; SWAG

Definitions

An *aggregation* is a function from a collection of data items to an aggregate value. In *sliding-*



Sliding-Window Aggregation Algorithms, Fig. 1
Sliding-window aggregation definitions

window aggregation, the input collection consists of a window over the most recent data items in a stream. Here, a *stream* is a potentially infinite sequence of data items, and the decision on which data items are *most recent* at any point in time is given by a window policy. A *sliding-window aggregation algorithm* updates the aggregate value, often using incremental-computation techniques, as the window contents change over time, as illustrated in Fig. 1.

Overview

Sliding-window aggregation summarizes a collection of recent streaming data, capturing the most recent happenings as well as some history. Including some history provides context for decisions, which would be missing if only the current data item were used. Using the most recent data helps identify and react to present trends, which would be diluted if all data from the beginning of time were included.

Aggregation is one of the most fundamental data processing operations. This is true in general, not just in stream processing. Aggregation is versatile: it can compute counts, averages, or maxima, index data structures, sketches such as Bloom filters, and many more.

In databases, it shows up as a basic relational algebra operator called group-by-aggregate and denoted γ (Garcia-Molina et al. 2008). In spreadsheets, it shows up as a function from a range of cells to a summary statistic (Sajaniemi and Pekkanen 1988). In programming languages, it shows up as a popular higher-order function called `fold` (Hutton 1999). In MapReduce, it shows up as `reduce` (Dean and Ghemawat 2004), which has been leveraged in many tasks, including computations that do not diminish the volume of data.

In stream processing, aggregation plays a similarly central role. But unlike the abovementioned cases, which focus on data at rest, streaming aggregation must handle data in motion. In particular, sliding-window aggregation must handle inserting new data items into the window as they arrive and evicting old data items from the window as they expire. Supporting this efficiently poses algorithmic challenges, especially for non-invertible aggregation functions such as `max`, for which there is no way to “subtract off” expiring items. From an algorithmic perspective, handling sliding windows with both insertion and eviction is more challenging than handling just insertion. Yet, there are two cases where eviction does not matter: unbounded and tumbling windows. Unbounded windows appear, for instance, in CQL (Arasu et al. 2006). Because they grow indefinitely, it is sufficient to update aggregations upon `insert` and not keep the data item itself around; they never need to call `evict`. Tumbling windows are more common; because they clear the entire contents of the window at the same time, there is no need to call `evict` on individual elements of the window.

Sliding windows are most commonly first-in, first-out (FIFO), resembling the behavior of a queue. What to keep in a sliding window and how often the aggregation is computed are controlled by policies, catalogued elsewhere (Gedik 2013); they may be count-based (e.g., the past 128 elements) or time-based (e.g., the past 12 min), among others. Regardless of policies, FIFO sliding-window aggregation (SWAG) can be formulated as an abstract data type with the following operations:

- `insert(v)` appends the value v to the window.
- `evict()` removes the “oldest” value in the window.
- `query()` returns the aggregation of the values in the window.

Metrics of interest in SWAG implementations are throughput, latency, and memory footprint. SWAG implementations also differ in generality: to enhance efficiency, aggregation operations, when feasible, are applied incrementally – that is, modifying a running sum of sort in response to data items arriving or leaving the window. To what extent this can be exploited depends on the nature of the aggregation operation.

Past work (Gray et al. 1996; Tangwongsan et al. 2015) cast most aggregation operations as binary operators, written \oplus , and has categorized them based on algebraic properties. Table 1 lists common aggregation operations with their properties and groups them into categories. An aggregation operator is *invertible* if there exists some function \ominus such that $(x \oplus y) \ominus y = x$ for all x and y . Using \ominus , SWAGs can implement eviction as an undo. A function is *associative* if $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ for all x , y , and z . SWAGs can take advantage of associativity by applying \oplus at arbitrary places inside the window. Without associativity, SWAGs are restricted to applying \oplus only at the end, upon insertion. A function is *commutative* if $x \oplus y = y \oplus x$ for all x and y . SWAGs are able to ignore the insertion order of data items for commutative aggregation operators. An aggregation operator is *rank-based* if it relies upon an ordering by some attribute of each data item, for instance, to find the i th-smallest.

Table 2 presents an overview of the SWAG algorithms presented in this article, with their asymptotic complexity, space usage, and restrictions. The most straightforward SWAG algorithm is called **Recalc**, since it always recalculates all values. Upon any `insert` or `evict`, Recalc walks the entire window and recomputes the aggregation value by using all available elements. Its performance is obviously $O(n)$, where n is

the current number of elements in the window. Recalc serves as the baseline comparison for all other SWAG algorithms. **Subtract-on-evict** (SOE) is a $O(1)$ algorithm, but it is not general: it can only be used when the aggregation is invertible. Upon every `insert`, SOE updates the current aggregation value using \oplus , and upon every `evict`, SOE updates that value using \ominus . The **order statistics tree** (OST) adds subtree statistics to the inner nodes of a balanced search tree (Hirzel et al. 2016). Values are put in both a queue and the tree, making both `insert` and `evict` $O(\log n)$. But `query` calls for aggregations such as the median or p th percentile become $O(\log n)$ because such information can be derived by traversing a path that is no longer than the height of the tree.

This section gave a brief overview with background and some simple aggregation algorithms. In general, research into SWAG algorithms tries to avoid $O(n)$ costs (unlike Recalc) but maintain generality (unlike SOE and OST). The next section will discuss the more sophisticated algorithms from Table 2 that offer improvements toward this goal.

Key Research Findings

The most successful techniques in speeding up sliding-window aggregation have been data structuring and algorithmic techniques that yield asymptotic improvements. They are the most effective when the aggregation function meets certain algebraic requirements. For instance, there are important aggregation operations that are associative, but not necessarily invertible nor commutative.

Pre-aggregation of data items that will be evicted at the same time is a technique that can be applied together with all SWAG algorithms discussed in this article. When data items are co-evicted, the window need not store them individually but can instead store partial aggregations, reducing the effective window size n in Table 2. Pre-aggregation algorithms include paned windows (Li et al. 2005), paired windows (Krishnamurthy et al. 2006), and Cutty

Sliding-Window Aggregation Algorithms, Table 1

Aggregation operations. Check marks (\checkmark), crosses (\times), and question marks (?) indicate that a property is true for all, false for all, or false for some of the given group, respectively

	Invertible	Associative	Commutative	Rank-based
• Sum-like : sum, count, average, standard deviation, ...	\checkmark	\checkmark	\checkmark	\times
• Collect-like : collect list, concatenate strings, i th-youngest, ...	\checkmark	\checkmark	\times	?
• Median-like : median, percentile, i th-smallest, ...	\checkmark	\checkmark	\checkmark	\checkmark
• Max-like : max, min, argMax, argMin, maxCount, ...	\times	\checkmark	?	\times
• Sketch-like : Bloom filter (Bloom 1970), CountMin (Cormode and Muthukrishnan 2005), HyperLogLog (Flajolet et al. 2007)	\times	\checkmark	\checkmark	\times

Sliding-Window Aggregation Algorithms, Table 2

Summary of aggregation algorithms and their properties, where n is the window size and n_{\max} is the size of the smallest contiguous range that contains all the shared windows

	Algorithmic complexity		Restrictions
	Time	Space	
Recalc	Worst-case $O(n)$	$O(n)$	None
Subtract-on-Evict (SOE)	Worst-case $O(1)$	$O(n)$	Sum-like or collect-like
Order Statistics Tree (OST) (Hirzel et al. 2016)	Worst-case $O(\log n)$	$O(n)$	Median-like
Reactive Aggregator (RA) (Tangwongsan et al. 2015)	Average $O(\log n)$	$O(n)$	Associative
DABA (Tangwongsan et al. 2017)	Worst-case $O(1)$	$O(n)$	Associative, FIFO
B-Int (Arasu and Widom 2004)	Shared $O(\log n_{\max})$	$O(n_{\max})$	Associative, FIFO
FlatFIT (Shein et al. 2017)	Average $O(1)$	$O(n)$	Associative, FIFO

windows (Carbone et al. 2016). Windows are sometimes coarsened to enable pre-aggregation, improving performance at the expense of some approximation.

B-Int (Arasu and Widom 2004), designed to facilitate sharing across windows, stores a “shared” window S that contains inside it all the windows being shared. To facilitate fast queries, B-Int maintains pre-aggregated values for all base intervals that lie within S . *Base intervals* (more commonly known now as dyadic intervals) are intervals of the form $[2^\ell k, 2^\ell(k+1)-1]$ with $\ell, k \geq 0$. The parameter ℓ defines the level of a base interval. This allows a query between the

i -th data item and j -th data item within S to be answered by combining at most $O(\log |i-j|)$ pre-aggregated values, resulting in logarithmic running time.

The **Reactive Aggregator** (RA) (Tangwongsan et al. 2015) is implemented via a balanced tree ordered by time, where internal nodes hold the partial aggregations of their subtrees, and offers $O(\log n)$ amortized time. Instead of the conventional approach to implementing balanced trees by frequent rebalancing, RA projects the tree over a complete perfect binary tree, which it stores in a flat array. This leads to higher performance than other

tree-based SWAG implementations in practice, since it saves the time of rebalancing as well as the overheads of pointers and fine-grained memory allocation.

For latency-sensitive applications, the aggregation algorithm cannot afford a long pause. **DABA** (Tangwongsan et al. 2017) ensures that every SWAG operation takes $O(1)$ time in the worst-case, not just on average. This is accomplished by extending Okasaki’s functional queue (Okasaki 1995) and removing dependencies on lazy evaluation and automatic garbage collection. **FlatFIT** (Shein et al. 2017) is another algorithm that achieves $O(1)$ time although in the amortized sense. This is accomplished by storing pre-aggregated values in a tree-like index structure that promotes reuse, reminiscent of path compression in the union-find data structure.

There are a number of other generic techniques that tend to apply broadly to sliding-window aggregation. Window partitioning is sometimes used as a means to maintain group-by aggregation and obtain data parallelism through fission (Schneider et al. 2015). When stream data items arrive out of order, a holding buffer can be used to reorder them before they enter the window (Srivastava and Widom 2004). Alternatively, in the case that the stream is formed by merging multiple sub-streams, out-of-order streams may be solved by pre-aggregating each data source separately and consolidating partial aggregation results as late as possible when doing an actual query (Krishnamurthy et al. 2010).

Examples of Application

Many applications of stream processing depend heavily upon sliding-window aggregation. This section describes concrete examples of applying sliding-window aggregation to real-world use cases. Understanding these examples helps appreciate the problems and guide the design of solutions.

Medical service providers want to save lives by getting early warnings when there is a high likelihood that a patient’s health is about to de-

teriorate. For instance, the Artemis system analyzes data from real-time sensors on patients in a neonatal intensive care unit (Blount et al. 2010). Among other things, it counts how often the blood oxygen saturation and the mean arterial blood pressure fall below a threshold in a 20-s sliding window. If the counts exceed another threshold, Artemis raises an alert.

Financial agents engaged in algorithmic trading want to make money by buying and selling stocks or other financial instruments. Treleaven et al. review the current practice for how that works technologically (Treleaven et al. 2013). Streaming systems for algorithmic trading make their decisions based on predicted future prices. One of the inputs for these predictions is a moving average of the recent history of a price, for example, over a 1-hour sliding window.

Road traffic can be regulated using variable tolling to implement congestion-pricing policies. One of the most popular benchmarks for streaming systems, linear road, is based on variable tolling (Arasu et al. 2004). The idea is to regulate demand by charging higher tolls for driving on congested roads. To do this, the streaming system must determine whether a road is congested. This works by using sliding-window aggregation to compute the number and average speed of vehicles in a given road segment and time window.

The above list of use cases is by no means exhaustive; there are many more applications of sliding-window aggregation, for instance, in phone providers, security, and social media.

Future Directions for Research

Research on sliding-window aggregation has focused mainly on aggregation functions that are associative and on FIFO windows. Much less is known for other nontrivial scenarios. Is it possible to efficiently support associative aggregation functions on windows that are non-FIFO? Besides associativity and invertibility, what other properties can be exploited to develop general-purpose algorithms for fast sliding-window aggregation? How can SWAG algorithms take better advantage of multicore parallelism?

Cross-References

- [Adaptive Windowing](#)
- [Incremental Sliding Window Analytics](#)
- [Stream Query Optimization](#)
- [Stream Window Aggregation Semantics and Optimization](#)

References

- Arasu A, Widom J (2004) Resource sharing in continuous sliding window aggregates. In: Conference on very large data bases (VLDB), pp 336–347
- Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Conference on very large data bases (VLDB), pp 480–491
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. *J Very Large Data Bases* 15(2):121–142
- Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. *Commun ACM* 13(7):422–426
- Blount M, Ebling MR, Eklund JM, James AG, McGregor C, Percival N, Smith K, Sow D (2010) Real-time analysis for intensive care: development and deployment of the Artemis analytic system. *IEEE Eng Med Biol Mag* 29:110–118
- Carbone P, Traub J, Katsifodimos A, Haridi S, Markl V (2016) Cutty: aggregate sharing for user-defined windows. In: Conference on information and knowledge management (CIKM), pp 1201–1210
- Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. *J Algorithms* 55(1):58–75
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Symposium on operating systems design and implementation (OSDI), pp 137–150
- Flajolet P, Fusy E, Gandouet O, Meunier F (2007) HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Conference on analysis of algorithms (AofA), pp 127–146
- Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Pearson/Prentice Hall, New Dehli
- Gedik B (2013) Generic windowing support for extensible stream processing systems. *Softw Pract Exp* 44(9):1105–1128
- Gray J, Bosworth A, Layman A, Pirahesh H (1996) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: International conference on data engineering (ICDE), pp 152–159
- Hirzel M, Rabbah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60
- Hutton G (1999) A tutorial on the universality and expressiveness of fold. *J Funct Program* 9(1):355–372
- Krishnamurthy S, Wu C, Franklin M (2006) On-the-fly sharing for streamed aggregation. In: International conference on management of data (SIGMOD), pp 623–634
- Krishnamurthy S, Franklin MJ, Davis J, Farina D, Golovko P, Li A, Thombre N (2010) Continuous analytics over discontinuous streams. In: International conference on management of data (SIGMOD), pp 1081–1092
- Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *ACM SIGMOD Rec* 34(1):39–44
- Okasaki C (1995) Simple and efficient purely functional queues and deques. *J Funct Program* 5(4):583–592
- Sajaniemi J, Pekkanen J (1988) An empirical analysis of spreadsheet calculation. *Softw Pract Exp* 18(6):583–596
- Schneider S, Hirzel M, Gedik B, Wu KL (2015) Safe data parallelism for general streaming. *IEEE Trans Comput* 64(2):504–517
- Shein AU, Chrysanthis PK, Labrinidis A (2017) FlatFIT: accelerated incremental sliding-window aggregation for real-time analytics. In: Conference on scientific and statistical database management (SSDBM), pp 5:1–5:12
- Srivastava U, Widom J (2004) Flexible time management in data stream systems. In: Principles of database systems (PODS), pp 263–274
- Tangwongsan K, Hirzel M, Schneider S, Wu KL (2015) General incremental sliding-window aggregation. In: Conference on very large data bases (VLDB), pp 702–713
- Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Conference on distributed event-based systems (DEBS), pp 66–77
- Treleaven P, Galas M, Lalchand V (2013) Algorithmic trading review. *Commun ACM* 56(11):76–85

S

Sliding-Window Fold

- [Sliding-Window Aggregation Algorithms](#)

Smart Cities or Future Cities

- [Big Data in Smart Cities](#)

Smart City Big Data or Large-Scale Urban Data and/or City Data

- ▶ [Big Data in Smart Cities](#)

Smart Industry

- ▶ [Big Data Warehouses for Smart Industries](#)

SnappyData

Barzan Mozafari
University of Michigan, Ann Arbor, MI, USA

Introduction

An increasing number of enterprise applications, particularly those in financial trading and IoT (Internet of Things), produce mixed workloads with all of the following: (1) continuous stream processing, (2) online transaction processing (OLTP), and (3) online analytical processing (OLAP). These applications need to simultaneously consume high-velocity streams to trigger real-time alerts, ingest them into a write-optimized transactional store, and perform analytics to derive deep insight quickly. Despite a flurry of data management solutions designed for one or two of these tasks, there is no single solution that is apt for all three.

SQL-on-Hadoop solutions (e.g., Hive, Impala/Kudu and SparkSQL) use OLAP-style optimizations and columnar formats to run OLAP queries over massive volumes of static data. While apt for batch processing, these systems are not designed as real-time operational

This article is based on Mozafari et al. (2017), authored by Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogen Mahajan, Soubhik Chakraborty, Hemant Bhanawat, Kishor Bachhav

databases, as they lack the ability to mutate data with transactional consistency, to use indexing for efficient point accesses, or to handle high-concurrency and bursty workloads.

Hybrid transaction/analytical processing (HTAP) systems, such as MemSQL, support both OLTP and OLAP queries by storing data in dual formats (row and columns) but need to be used alongside an external streaming engine (e.g., Storm (Toshniwal et al. 2014), Kafka, Confluent) to support stream processing. They also lack approximation features required for interactive-speed analytics or visualization workloads (Park et al. 2016).

Finally, there are numerous academic (Chandrasekaran et al. 2003; Mozafari et al. 2012; Thakkar et al. 2011) and commercial (Apache Samza; Toshniwal et al. 2014; TIBCO; Akidau et al. 2013) solutions for stream and event processing. Although some stream processors provide some form of state management or transactions (e.g., Samza (Apache Samza), Liquid (Fernandez et al. 2015), S-Store (Meehan et al. 2015)), they only allow simple queries on streams. However, more complex analytics, such as joining a stream with a large history table, need the same optimizations used in an OLAP engine (Liarou et al. 2012; Braun et al. 2015; Thakkar et al. 2011). For example, streams in IoT are continuously ingested and correlated with large historical data. Trill (Chandramouli et al. 2014) supports diverse analytics on streams and columnar data but lacks transactions. DataFlow (Akidau et al. 2015) focuses on logical abstractions rather than a unified query engine.

Consequently, the demand for mixed workloads has resulted in several composite data architectures, exemplified in the “lambda” architecture, which requires multiple solutions to be stitched together – a difficult exercise that is time-consuming and expensive.

In capital markets, for example, a real-time market surveillance application has to ingest trade streams at very high rates and detect abusive trading patterns (e.g., insider trading). This requires correlating large volumes of data by joining a stream with (1) historical records, (2) other streams, and (3) financial reference

data which can change throughout the trading day. A triggered alert could in turn result in additional analytical queries, which will need to run on both ingested and historical data. In this scenario, trades arrive on a message bus (e.g., Tibco, IBM MQ, Kafka) and are processed by a stream processor (e.g., Storm) or a homegrown application, while the state is written to a key-value store (e.g., Cassandra) or an in-memory data grid (e.g., GemFire). This data is also stored in HDFS and analyzed periodically using a SQL-on-Hadoop or a traditional OLAP engine.

These heterogeneous workflows, although far too common in practice, have several drawbacks (D1–D4):

D1. Increased complexity and total cost of ownership: The use of incompatible and autonomous systems significantly increases their total cost of ownership. Developers have to master disparate APIs, data models, and tuning options for multiple products. Once in production, operational management is also a nightmare. To diagnose the root cause of a problem, highly paid experts spend hours to correlate error logs across different products.

D2. Lower performance: Performing analytics necessitates data movement between multiple non-colocated clusters, resulting in several network hops and multiple copies of data. Data may also need to be transformed when faced with incompatible data models (e.g., turning Cassandra's ColumnFamilies into Storm's domain objects).

D3. Wasted resources: Duplication of data across different products wastes network bandwidth (due to increased data shuffling), CPU cycles, and memory.

D4. Consistency challenges: The lack of a single data governance model makes it harder to reason about consistency semantics. For instance, a lineage-based recovery in Spark Streaming may replay data from the last checkpoint and ingest

it into an external transactional store. With no common knowledge of lineage and the lack of distributed transactions across these two systems, ensuring exactly once semantics is often left as an exercise for the application ([Exactly-once processing with trident](#)).

Challenges

SnappyData's goal is to reduce complexity and improve performance by offering streaming, transaction processing, and interactive analytics in a single cluster (Ramnarayan et al. 2016). Realizing this goal involves overcoming several challenges. The first challenge is the drastically different data structures and query processing paradigms that are optimal for each type of workload. For example, column stores are optimal for analytics, transactions need write-optimized row-stores, and infinite streams are best handled by sketches and windowed data structures. Likewise, while analytics thrive with batch processing, transactions rely on point lookups/updates, and streaming engines use delta/incremental query processing. Marrying these conflicting mechanisms in a single system is challenging, as is abstracting away this heterogeneity from programmers.

Another challenge is the difference in expectations of high availability (HA) across different workloads. Scheduling and resource provisioning are also harder in a mixed workload of streaming jobs, long-running analytics, and short-lived transactions. Finally, achieving interactive analytics becomes nontrivial when deriving insight requires joining a stream against large historical data ([Makin' Bacon and the Three Main Classes of IoT Analytics](#)).

Approach Overview

To support mixed workloads, SnappyData carefully fuses Apache Spark, as a computational engine, with Apache GemFire, as a transactional store.

Through a common set of abstractions, Spark allows programmers to tackle a confluence of different paradigms (e.g., streaming, machine learning, SQL analytics). Spark's core abstraction, a

Resilient Distributed Dataset (RDD), provides fault tolerance by efficiently storing the lineage of all transformations instead of the data. The data itself is partitioned across nodes and if any partition is lost, it can be reconstructed using its lineage. The benefit of this approach is twofold: avoiding replication over the network and higher throughput by operating on data as a batch. While this approach provides efficiency and fault tolerance, it also requires that an RDD be immutable. In other words, Spark is simply designed as a computational framework and therefore (i) does not have its own storage engine and (ii) does not support mutability semantics. (Although `IndexedRDD` ([Indexedrdd for apache spark](#)) offers an updatable key-value store ([Indexedrdd for apache spark](#)), it does not support colocation for high-rate ingestions or distributed transactions. It is also unsuitable for HA, as it relies on disk-based checkpoints for fault tolerance.)

On the other hand, Apache `GemFire` ([Apache Geode](#)) (a.k.a. Geode) is one of the most widely adopted in-memory data grids in the industry, which manages records in a partitioned row-oriented store with synchronous replication. It ensures consistency by integrating a dynamic group membership service and a distributed transaction service. GemFire allows for indexing and both fine-grained and batched data updates. Updates can be reliably enqueued and asynchronously written back out to an external database. In-memory data can also be persisted to disk using

append-only logging with offline compaction for fast disk writes ([Apache Geode](#)).

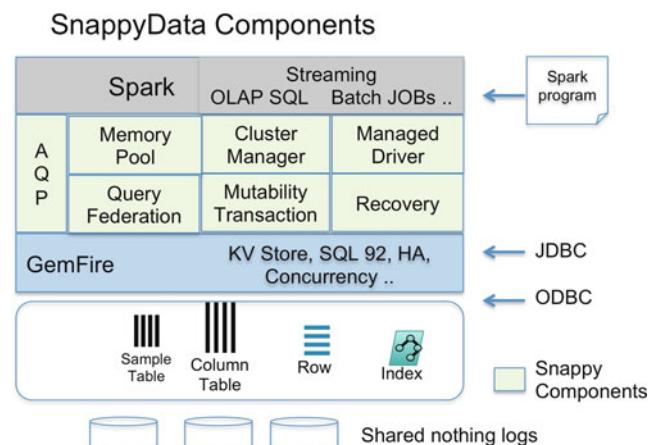
Best of two worlds – To combine the best of both worlds, SnappyData seamlessly integrates Spark and GemFire runtimes, adopting Spark as the programming model with extensions to support mutability and HA (high availability) through GemFire’s replication and fine-grained updates. This marriage, however, poses several nontrivial challenges. For instance, when ingesting a stream, SnappyData processes the incoming stream as a batch, avoids replication, and replays from the source on a failure. To avoid a tuple-at-a-time replication, the processed state can be written to the store in batches. Recovery from failure will thus be limited to the time needed to replay a single batch.

Architecture

Figure 1 depicts SnappyData’s core components (the original components from Spark and GemFire are highlighted).

SnappyData’s hybrid storage layer is primarily in-memory and can manage data in row, column, or probabilistic stores. SnappyData’s column format is derived from Spark’s RDD implementation. SnappyData’s row-oriented tables extend GemFire’s table and thus support indexing and fast reads/writes on indexed keys. In addition to

SnappyData, Fig. 1
SnappyData’s core components



these “exact” stores, SnappyData can also summarize data in *probabilistic* data structures, such as stratified samples and other forms of synopses. SnappyData’s query engine has built-in support for approximate query processing (AQP), which can exploit these probabilistic structures. This allows applications to trade accuracy for interactive-speed analytics on streams or massive datasets.

SnappyData supports two programming models – SQL (by extending SparkSQL dialect) and Spark’s API. Thus, one can perceive SnappyData as a SQL database that uses Spark’s API as its language for stored procedures. Stream processing in SnappyData is primarily through Spark Streaming, but it is modified to run in situ with SnappyData’s store.

SQL queries are federated between Spark’s Catalyst and GemFire’s OLTP engine. An initial query plan determines if the query is a low-latency operation (e.g., a key-based lookup) or a high-latency one (scans/aggregations). SnappyData avoids scheduling overheads for OLTP operations by immediately routing them to appropriate data partitions.

To support replica consistency, fast point updates, and instantaneous detection of failure conditions in the cluster, SnappyData relies on GemFire’s P2P (peer-to-peer) cluster membership service ([Apache Geode](#)). Transactions follow a two-phase commit protocol using GemFire’s Paxos implementation to ensure consensus and view consistency across the cluster.

A Unified API

Spark offers a rich procedural API for querying and transforming disparate data formats (e.g., JSON, Java Objects, CSV). Likewise, to retain a consistent programming style, SnappyData offers its mutability functionalities as extensions of SparkSQL’s dialect and its DataFrame API. These extensions are backward compatible, i.e., applications that do not use them observe Spark’s original semantics.

A DataFrame in Spark is a distributed collection of data organized into named columns. A DataFrame can be accessed from

a SQLContext, which itself is obtained from a SparkContext (a SparkContext is a connection to Spark’s cluster). Likewise, much of SnappyData’s API is offered through SnappyContext, which is an extension of SQLContext. Listing 1 is an example of using SnappyContext.

```

1 // Create a SnappyContext from a SparkContext
2 val spContext = new org.apache.spark.
3   SparkContext(conf)
4 val sncContext = org.apache.spark.sql.
5   SnappyContext(spContext)
6
7 // Create a column table using SQL
8 sncContext.sql("CREATE TABLE MyTable (id int,
9   data string) using column")
10
11 // Append contents of a DataFrame
12   into the table
13 someDataDF.write.insertInto("MyTable");
14
15 // Access the table as a DataFrame
16 val myDataFrame: DataFrame = sncContext.
17   table("MyTable")
18 println(s"Number of rows in MyTable = {
19   myDataFrame.count()}")

```

Listing 1 Working with DataFrames in SnappyData

Stream processing often involves maintaining counters or more complex multidimensional summaries. As a result, stream processors today are either used alongside a scale-out in-memory key-value store (e.g., Storm with Redis or Cassandra) or come with their own basic form of state management (e.g., Samza, Liquid (Fernandez et al. 2015)). These patterns are often implemented in the application code using simple get/put APIs. While these solutions scale well, most users tend to modify their search patterns and trigger rules quite often. These modifications require expensive code changes and lead to brittle and hard-to-maintain applications.

In contrast, SQL-based stream processors offer a higher-level abstraction to work with streams but primarily depend on row-oriented stores (e.g., [IBM](#); [TIBCO](#); Meehan et al. (2015)) and are thus limited in supporting complex analytics. To support continuous queries with scans, aggregations, top-K queries, and joins with historical and reference data, some of the

same optimizations found in OLAP engines must be incorporated in the streaming engine (Liarou et al. 2012). Thus, SnappyData extends Spark Streaming to allow declaring and querying streams in SQL. More importantly, SnappyData provides OLAP-style optimizations to enable scalable stream analytics, including columnar formats, approximate query processing, and co-partitioning (SnappyData 2016).

Hybrid Store: Row and Column Tables

Tables can be partitioned or replicated and are primarily managed in memory with one or more consistent replicas. The data can be managed in Java heap memory or off-heap. Partitioned tables are always partitioned horizontally across the cluster. For large clusters, SnappyData allows data servers to belong to one or more logical groups, called “server groups.” The storage format can be “row” (either partitioned or replicated tables) or “column” (only supported for partitioned tables) format. Row tables incur a higher in-memory footprint but are well suited to random updates and point lookups, especially with in-memory indexes. Column tables manage column data in contiguous blocks and are compressed using dictionary, run-length, or bit encoding (Xin and Rosen).

SnappyData extends Spark’s column store to support mutability. Updating row tables is trivial. When records are written to column tables, they first arrive in a *delta row buffer* that is capable of high write rates and then age into a columnar form. The delta row buffer is merely a partitioned row table that uses the same partitioning strategy as its base column table. This buffer table is backed by a conflating queue that periodically empties itself as a new batch into the column table. Here, conflation means that consecutive updates to the same record result in only the final state getting transferred to the column store. For example, inserted/updated records followed by deletes are removed from the queue. The delta row buffer itself uses copy-on-write semantics to ensure that concurrent application updates do not cause inconsistency (Abadi et al. 2013). SnappyData extends Spark’s Catalyst optimizer

to merge the delta row buffer during query execution.

Probabilistic Store

Achieving interactive response time is challenging when running complex analytics on streams, e.g., joining a stream with a large table (Mozafari and Zaniolo 2010). Even OLAP queries on stored datasets can take tens of seconds to complete if they require a distributed shuffling of records or if hundreds of concurrent queries run in the cluster (Agarwal et al. 2012). In such cases, SnappyData’s storage engine is capable of using probabilistic structures to dramatically reduce the volume of input data and provide approximate but extremely fast answers. In this regard, SnappyData can be seen as the first full-fledged commercial AQP engine (see Mozafari (2017) for why the adoption of AQP has not previously slowed). SnappyData’s probabilistic structures include uniform samples, stratified samples, and sketches (Mozafari and Niu 2015). Unlike VerdictDB (Park et al. 2018; He et al. 2018) and Database Learning (Park et al. 2017), which are agnostic of the underlying query engine, SnappyData’s probabilistic store is tightly integrated into its query processing logic. SnappyData’s approach is also different from other AQP engines (Zeng et al. 2014a; Agarwal et al. 2012; Zeng et al. 2014b), in the way that it creates and maintains these structures efficiently and in a distributed manner. However, given these structures, SnappyData uses off-the-shelf error estimation techniques (Agarwal et al. 2014). SnappyData’s sample selection and maintenance strategies are discussed next.

Sample selection – Unlike uniform samples, choosing which stratified samples to build is a nontrivial problem. The key question is which sets of columns to build a stratified sample on. Prior work has used skewness, popularity, and storage cost as the criteria for choosing column sets (Agarwal et al. 2012, 2013). SnappyData extends these criteria as follows: for any declared or foreign-key join, the join key is included in a stratified sample in at least one of the participating relations (tables or streams). However,

SnappyData never includes a table’s primary key in its stratified sample(s). Furthermore, SnappyData uses an open-source tool, called `WorkloadMiner`, which automatically analyzes past query logs and reports a rich set of statistics ([CliffGuard](#)). These statistics guide SnappyData’s users through the sample selection process. `WorkloadMiner` is integrated into `CliffGuard`. `CliffGuard` guarantees a robust physical design (e.g., set of samples), which remains optimal even if future queries deviate from past ones ([Mozafari et al. 2015](#)).

Once a set of samples is chosen, the challenge is how to update them, which is a key differentiator between SnappyData and previous AQP systems that use stratified samples ([Chaudhuri et al. 2007](#); [Agarwal et al. 2013](#); [Zeng et al. 2015](#)).

Sample maintenance – Previous AQP engines that use offline sampling update and maintain their samples periodically using a single scan of the entire data ([Mozafari and Niu 2015](#)). This strategy is not suitable for SnappyData with streams and mutable tables for two reasons. First, maintaining per-stratum statistics across different nodes in the cluster is a complex process. Second, updating a sample in a streaming fashion requires maintaining a reservoir ([Vitter et al. 1985](#); [Al-Kateb and Lee 2010](#)), which means the sample must either fit in memory or be evicted to disk. Keeping samples entirely in memory is impractical for infinite streams unless the sampling rate is perpetually decreased. Likewise, disk-based reservoirs are inefficient as they require retrieving and removing individual tuples from disk as new tuples are sampled.

To solve these problems, SnappyData always includes timestamp as an additional column in every stratified sample. Uniform samples are treated as a special case with only one stratified column, i.e., timestamp. As new tuples arrive in a stream, a new batch (in row format) is created for maintaining a sample of each observed value of the stratified columns. Whenever a batch size exceeds a certain threshold (1M tuples by default), it is evicted and archived to disk (in a columnar format), and a new batch is started for that stratum.

Treating each micro-batch as an independent stratified sample has several benefits. First, this allows SnappyData to adaptively adjust the sampling rate for each micro-batch without the need for internode communications in the cluster. Second, once a micro-batch is completed, its tuples never need to be removed or replaced, and therefore they can be safely stored in a compressed columnar format and even archived to disk. Only the latest micro-batch needs to be in-memory and in row format. Finally, each micro-batch can be routed to a single node, reducing the need for network shuffles.

State Sharing

SnappyData hosts GemFire’s tables in the executor nodes as either partitioned or replicated tables. When partitioned, the individual buckets are presented as Spark RDD partitions, and their access is therefore parallelized. This is similar to the way that any external data source is accessed in Spark, except that the common operators are optimized in SnappyData. For example, by keeping each partition in columnar format, SnappyData avoids additional copying and serialization and speeds up scan and aggregation operators. SnappyData can also colocate tables by exposing an appropriate partitioner to Spark.

Native Spark applications can register any DataFrame as a temporary table. In addition to being visible to the Spark application, such a table is also registered in SnappyData’s catalog – a shared service that makes tables visible across Spark and GemFire. This allows remote clients connecting through ODBC/JDBC to run SQL queries on Spark’s temporary tables as well as tables in GemFire.

In streaming scenarios, the data can be sourced into any table from parent stream RDDs (DStream), which themselves could source events from an external queue, such as Kafka. To minimize shuffling, SnappyData tables can preserve the partitioning scheme used by their parent RDDs. For example, a Kafka queue listening on Telco CDRs (call detail records) can be partitioned on `subscriberID` so that Spark’s DStream and the SnappyData table ingesting these records will be partitioned on the same key.

Locality-Aware Partition Design

A major challenge in horizontally partitioned distributed databases is to restrict the number of nodes involved in order to minimize (i) shuffling during query execution and (ii) distributed locks (Helland 2007; Zamanian et al. 2015). In addition to network costs, shuffling can also cause CPU bottlenecks by incurring excessive copying (between kernel and user space) and serialization costs (Ousterhout et al. 2015). To reduce the need for shuffling and distributed locks, SnappyData’s data model promotes two fundamental ideas:

- 1. Co-partitioning with shared keys** – A common technique in data placement is to take the application’s access patterns into account. SnappyData pursues a similar strategy: since joins require a shared key, it co-partitions related tables on the join key. SnappyData’s query engine can then optimize its query execution by localizing joins and pruning unnecessary partitions.
- 2. Locality through replication** – Star schemas are quite prevalent, wherein a few ever-growing fact tables are related to several dimension tables. Since dimension tables are relatively small and change less often, schema designers can ask SnappyData to replicate these tables. SnappyData particularly uses these replicated tables to optimize joins.

Dynamic rebalancing of data – When access is non-uniformly distributed across the keys, a load imbalance occurs where a few servers end up performing most of the work. For instance, when tracking users’ browsing behavior on a website, a few popular pages will dominate the rest. SnappyData provides metrics on which nodes are being accessed heavily and also provides administrative APIs that can be used to move “hot buckets” of data to a different node. If the imbalance is a memory usage imbalance, admin APIs can be used to trigger a rebalance which is a non blocking operation that moves buckets of data to less loaded nodes in the background and restores memory balance. Used effectively,

rebalancing prevents hotspots from developing in the system and avoid performance bottlenecks.

Hybrid Cluster Manager

Spark applications run as independent processes in the cluster, coordinated by the application’s main program, called the driver program. Spark applications connect to cluster managers (YARN or Mesos) to acquire executor nodes. While Spark’s approach is appropriate for long-running tasks, as an operational database, SnappyData’s cluster manager must meet additional requirements, such as high concurrency, high availability, and consistency.

High Availability

To ensure high availability (HA), SnappyData needs to detect faults and be able to recover from them instantly.

Failure detection – Spark uses heartbeat communications with a central master process to determine the fate of the workers. Since Spark does not use a consensus-based mechanism for failure detection, it risks shutting down the entire cluster due to master failures. However, as an always-on operational database, SnappyData needs to detect failures faster and more reliably. For faster detection, SnappyData relies on UDP neighbor ping and TCP ack timeout during normal data communications. To establish a new, consistent view of the cluster membership, SnappyData relies on GemFire’s weighted quorum-based detection algorithm ([Apache Geode](#)). Once GemFire establishes that a member has indeed failed, it ensures that a consistent view of the cluster is applied to all members, including the Spark master, driver, and data nodes.

Failure recovery – Recovery in Spark is based on logging the transformations used to build an RDD (i.e., its lineage) rather than the actual data. If a partition of an RDD is lost, Spark has sufficient information to recompute just that partition (Zaharia et al. 2012). Spark can also checkpoint RDDs to stable storage to shorten

the lineage, thereby shortening the recovery time. The decision of when to checkpoint, however, is left to the user. GemFire, on the other hand, relies on replication for instantaneous recovery but at the cost of lower throughput. SnappyData merges these recovery mechanisms as follows:

1. Fine-grained updates issued by transactions avoid the use of Spark’s lineage altogether and instead use GemFire’s eager replication for fast recovery.
2. Batched and streaming micro-batch operations are still recovered by RDD’s lineage, but instead of HDFS, SnappyData writes their checkpoints to GemFire’s in-memory storage, which itself relies on a fast P2P (peer-to-peer) replication for recovery. Also, SnappyData’s intimate knowledge of the load on the storage layer, the data size, and the cost of recomputing a lost partition allows for automating the choice of checkpoint intervals based on an application’s tolerance for recovery time.

Hybrid Scheduler and Provisioning

Thousands of concurrent clients can simultaneously connect to a SnappyData cluster. To support this degree of concurrency, SnappyData categorizes incoming requests as low- and high-latency operations. By default, SnappyData treats a job as a low-latency operation unless it accesses a columnar table. However, applications can also explicitly label their latency sensitivity. SnappyData allows low-latency operations to bypass Spark’s scheduler and directly operate on the data. High-latency operations are passed through Spark’s fair scheduler. However, among the low-latency operations, SnappyData still relies on a simple FIFO policy (other systems, such as MariaDB or MySQL, use more sophisticated algorithms for transaction scheduling, e.g., VATS (Huang et al. 2017) or MySQL’s CATS (Tian et al. 2018)). For low-latency operations, SnappyData attempts to reuse their executors to maximize their data locality (in-process). For high-latency jobs, SnappyData dynamically expands their compute resources while retaining the nodes caching their data.

Consistency Model

SnappyData relies on GemFire for its consistency model. GemFire supports “read committed” and “repeatable read” transaction isolation levels using a variant of the Paxos algorithm (Gray and Lamport 2006). Transactions detect write-write conflicts and assume that writers rarely conflict. When write locks cannot be obtained, transactions abort without blocking ([Apache Geode](#)).

SnappyData extends Spark’s `SparkContext` and `SQLContext` to add mutability semantics. SnappyData gives each SQL connection its own `SQLContext` in Spark to allow applications to start, commit, and abort transactions.

While any RDD obtained by a Spark program observes a consistent view of the database, multiple programs can observe different views when transactions interleave. An MVCC mechanism (based on GemFire’s internal row versions) can be used to deliver a single snapshot view to the entire application.

In streaming applications, upon faults, Spark recovers lost RDDs from their lineage. This means that some subset of the data will be replayed. To cope with such cases, SnappyData ensures the exactly once semantics at the storage layer so that multiple write attempts are idempotent, hence relieving developers of having to ensure this in their own applications. SnappyData achieves this goal by placing the entire flow as a single transactional unit of work, whereby the source (e.g., a Kafka queue) is acknowledged only when the micro-batch is entirely consumed and the application state is successfully updated. This ensures automatic rollback of incomplete transactions.

S

Conclusion

SnappyData is a unified platform for real-time operational analytics, which supports OLTP, OLAP, and stream analytics in a single integrated solution. SnappyData’s approach is a deep integration of a computational engine for high-throughput analytics (Spark) with a scale-out in-memory transactional store (GemFire). SnappyData extends SparkSQL and Spark Streaming APIs

with mutability semantics and offers various optimizations to enable colocated processing of streams and stored datasets. SnappyData has integrated approximate query processing for enabling real-time operational analytics over large (stored or streaming) data. Overall, SnappyData's goal is to yield a significantly lower TCO for mixed workloads compared to using disparate products that are managed, deployed, and monitored separately.

References

- Abadi D et al (2013) The design and implementation of modern column-oriented database systems. Found Trends Databases 5(3):197–280
- Agarwal S, Panda A, Mozafari B, Iyer AP, Madden S, Stoica I (2012) Blink and it's done: interactive queries on very large data. In: PVLDB
- Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: EuroSys
- Agarwal S, Milner H, Kleiner A, Talwalkar A, Jordan M, Madden S, Mozafari B, Stoica I (2014) Knowing when you're wrong: building fast and reliable approximate query processing systems. In: SIGMOD
- Akida T et al (2013) MillWheel: fault-tolerant stream processing at internet scale. In: PVLDB
- Akida T et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In: PVLDB
- Al-Kateb M, Lee BS (2010) Stratified reservoir sampling over heterogeneous data streams. In: SSDBM
- Apache Geode. <http://geode.incubator.apache.org/>
- Apache Samza. <http://samza.apache.org/>
- Barber R, Huras M, Lohman G, Mohan C, Mueller R, Özcan F, Pirahesh H, Raman V, Sidle R, Sidorkin O et al (2016) Wildfire: concurrent blazing data ingest and analytics. In: SIGMOD
- Braun L et al (2015) Analytics in motion: high performance event-processing and real-time analytics in the same database. In: SIGMOD
- Chandramouli B et al (2014) Trill: a high-performance incremental query processor for diverse analytics. In: PVLDB
- Chandrasekaran S et al (2003) TelegraphCQ: continuous dataflow processing. In: SIGMOD
- Chaudhuri S, Das G, Narasayya V (2007) Optimized stratified sampling for approximate query processing. ACM Trans Database Syst 32(2):9
- CliffGuard. A general framework for robust and efficient database optimization. <http://www.cliffguard.org>
- Exactly-once processing with trident – the fake truth. <https://www.alooma.com/blog/trident-exactly-once>
- Fernandez RC et al (2015) Liquid: unifying nearline and offline big data integration. In: CIDR
- Gray J, Lamport L (2006) Consensus on transaction commit. ACM Trans Database Syst 31(1): 133–160
- He W, Park Y, Hanafi I, Yatvitskiy J, Mozafari B (2018) Demonstration of VerdictDB, the platform-independent AQP system. In: SIGMOD
- Helland P (2007) Life beyond distributed transactions: an apostate's opinion. In: CIDR
- Huang J, Mozafari B, Schoenebeck G, Wenisch T (2017) A top-down approach to achieving performance predictability in database systems. In: SIGMOD
- IBM. InfoSphere BigInsights. <http://tinyurl.com/ouphdss>
- Indexedrdd for apache spark. <https://github.com/amplab/spark-indexedrdd>
- Liarou E et al (2012) Monetdb/datacell: online analytics in a streaming column-store. In: PVLDB
- Makin' Bacon and the Three Main Classes of IoT Analytics. <http://tinyurl.com/zlc6den>
- Meehan J et al (2015) S-store: streaming meets transaction processing. In: PVLDB
- Mozafari B (2017) Approximate query engines: commercial challenges and research opportunities. In: SIGMOD
- Mozafari B, Zaniolo C (2010) Optimal load shedding with aggregates and mining queries. In: ICDE
- Mozafari B, Niu N (2015) A handbook for building an approximate query engine. IEEE Data Eng Bull 38(3):3–29
- Mozafari B, Zeng K, Zaniolo C (2012) High-performance complex event processing over xml streams. In: SIGMOD
- Mozafari B, Ye Goh EZ, Yoon DY (2015) CliffGuard: a principled framework for finding robust database designs. In: SIGMOD
- Mozafari B, Ramnarayan J, Menon S, Mahajan Y, Chakraborty S, Bhanawat H, Bachhav K (2017) SnappyData: a unified cluster for streaming, transactions, and interactive analytics. In: CIDR
- Ousterhout K et al (2015) Making sense of performance in data analytics frameworks. In: NSDI
- Park Y, Cafarella M, Mozafari B (2016) Visualization-aware sampling for very large databases. In: ICDE
- Park Y, Tajik AS, Cafarella M, Mozafari B (2017) Database learning: towards a database that becomes smarter every time. In: SIGMOD
- Park Y, Mozafari B, Sorenson J, Wang J (2018) VerdictDB: universalizing approximate query processing. In: SIGMOD
- Ramnarayan J, Mozafari B, Menon S, Wale S, Kumar N, Bhanawat H, Chakraborty S, Mahajan Y, Mishra R, Bachhav K (2016) SnappyData: a hybrid transactional analytical store built on spark. In: SIGMOD
- SnappyData (2016) Streaming, transactions, and interactive analytics in a unified engine. <http://web.eecs.umich.edu/~mozafari/php/data/uploads/snappy.pdf>

- Thakkar H, Laptev N, Mousavi H, Mozafari B, Russo V, Zaniolo C (2011) SMM: a data stream management system for knowledge discovery. In: ICDE TIBCO. StreamBase. <http://www.streambase.com/>
- Tian B, Huang J, Mozafari B, Schoenebeck G, Wenisch T (2018) Contention-aware lock scheduling for transactional databases. In: PVLDB
- Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm@twitter. In: SIGMOD
- Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57
- Xin R, Rosen J. Project Tungsten: bringing Spark closer to bare metal. <http://tinyurl.com/mzw7hew>
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI
- Zamanian E, Binnig C, Salama A (2015) Locality-aware partitioning in parallel database systems. In: SIGMOD
- Zeng K, Gao S, Gu J, Mozafari B, Zaniolo C (2014a) ABS: a system for scalable approximate queries with accuracy guarantees. In: SIGMOD
- Zeng K, Gao S, Mozafari B, Zaniolo C (2014b) The analytical bootstrap: a new method for fast error estimation in approximate query processing. In: SIGMOD
- Zeng K, Agarwal S, Dave A, Armbrust M, Stoica I (2015) G-OLA: generalized on-line aggregation for interactive analysis on big data. In: SIGMOD

Social Networking

- Big Data in Social Networks

Spark SQL

Xiao Li, Cheng Lian, and Shu Mo
Databricks Inc., San Francisco, CA, USA

Definitions

SQL is a highly scalable and efficient relational processing engine with ease-to-use APIs and mid-query fault tolerance. It is a core module of Apache Spark, which is a unified engine for distributed data processing (Zaharia et al. 2012). Spark SQL can process, integrate, and analyze the data from diverse data sources (e.g.,

Hive, Cassandra, Kafka, and Oracle) and file formats (e.g., Parquet, ORC, CSV, and JSON). The common use cases include ad hoc analysis, logical warehouse, query federation, and ETL processing. It also powers the other Spark libraries, including structured streaming for stream processing, MLlib for machine learning (Meng et al. 2016; Michael et al. 2018), GraphFrame for graph-parallel computation (Dave et al. 2016), and TensorFrames for TensorFlow binding. These libraries and Spark SQL can be seamlessly combined in the same application with holistic optimization by Spark SQL.

Overview

Spark is a general purpose big data processing system. It was originally developed in the AMPLab at UC Berkeley and donated to Apache Software Foundation in 2013. Now, Apache Spark is one of the most popular open-source projects in data analytics and query processing. Spark SQL (Armbrust et al. 2015) (its predecessor, Shark (Xin et al. 2013)) was introduced in 2014 and became the core of Apache Spark ecosystem. It enables Spark to perform efficient and fault-tolerant relational query processing with analytics database technologies. The relational queries are compiled to Resilient Distributed Dataset (RDD) transformations and actions, which are executable in Spark.

Spark SQL follows the classic query processing and federation architecture with adoption of the recent research (e.g., whole-stage code generation). Through the user-facing APIs (SQL, DataFrame, and Dataset), the user queries are converted to unresolved abstract syntax trees (called unresolved logical plans). The plans are then analyzed using the session-specific temporary view manager, and the cross-session cache manager and catalog. Logical optimizer and physical planner optimize the resolved plans by applying heuristics-based and cost-based transformation rules. During the query planning phase, the sub-plans are pushed down to the underlying data sources for more

efficient processing, if possible; the logical plans are converted to the executable physical plans consisting of transformations and actions on RDDs with the generated Java code. The code is compiled to Java bytecode, executed at runtime by JVM and optimized by JIT to native machine code at runtime.

Declarative APIs

Prior to Spark SQL, Spark offers low-level procedural APIs for operating RDDs and building DAGs that are executable in Spark. Spark SQL introduces three declarative APIs (DataFrame, Dataset, SQL language), which are complementary to the low-level Spark APIs (i.e., RDD APIs). It facilitates tight integration of relational queries and complex procedural processing.

The SQL API is based on ANSI SQL:2003 standard with full compliance to Hive query language (HiveQL). For the existing Hive users, the compliance facilitates migration to Spark SQL. Spark SQL also provides language-integrated and lazily evaluated DataFrame-/Dataset APIs. The DataFrame API provides untyped relational operations, while the Dataset API provides a typed version for better type safety. The DataFrame API is available in Scala, Java, Python, and R. The Dataset API is only available in Scala and Java since Python and R are dynamically typed languages.

Compared with SQL, DataFrame/Dataset APIs provide richer and user-friendly interfaces, since the APIs are not limited by ANSI SQL compliance and also fully integrated with the programming languages (Java/Scala/Python/R). Using DataFrame/Dataset APIs, a complex data-flow logic can be split to multiple simpler modular host-language functions and then build up a single logical plan for holistic query optimization. All these three APIs are internally represented by the same Catalyst logical plans. They can be mixed, combined, optimized, and executed holistically, thanks to the lazy execution. The execution is triggered until users call the action APIs (e.g., collect, save, and show).

Query Optimization

Spark SQL optimizes the plans using the stratified search strategy that are widely used in the commercial database vendors (e.g., Oracle and DB2). First, the optimizer rewrites the query plans using heuristics-based rules. The typical transformation rules include predicate pushdown, column pruning, outer join elimination, constraint propagation, and predicate inference. Then, the cost-based plan enumeration and method selection are executed. The cost-based optimizer framework is initially shipped with Spark 2.2 and still rapidly evolving. Histogram was introduced for cardinality estimation in Spark 2.3. More accurate cost model, demand-driven enumerators, and adaptive query optimization are in the road map.

Spark SQL optimizer is extensible. Custom optimizer rules can be injected. For example, the users can add extra optimization rules for pushing more operators into the external data source systems or supporting the new data types.

Query Execution

Memory and CPU, rather than disk and network I/O, are the major performance bottlenecks of query execution in Apache Spark (Ousterhout et al. 2017), thanks to the progress in related hardware and data compression. The project Tungsten speeds up query execution by optimizing the efficiency of CPU and memory. The major focuses include off-heap memory management and runtime code generation.

Memory Management

The overhead of JVM objects and GC are significant for data intensive Java applications. Apache Spark leverages the application semantics to explicitly manage the memory using the *sun.misc.Unsafe* feature. This feature provides the C-style direct access to off-heap memory. The memory managed by Spark is invisible to the garbage collector. Tuning

GC for higher performance is not needed. The compiled/interpreted operations directly manipulate the binary data represented in the Tungsten specialized format. Compared with JVM objects, it has less memory footprint and thus reduces the processing time.

Runtime Code Generation

On the driver, Spark generates Java source code specialized to each query. On the executors, the generated code is compiled to Java bytecode by a lightweight Java compiler, which runs directly on the JVM and can be further compiled to native code by the just-in-time (JIT) compiler in the JVM. It strikes a good balance between performance and readability (and thus debuggability) of generated code.

Runtime code generation is performed on two levels: (1) expressions are generated into straightforward Java code to reduce interpretation overhead, and (2) where possible, multiple adjacent physical plan operators, along with the expressions involved, are fused together using a push-based model and generated into a single code generation unit (called a stage). Compared with the original iterator-based pull model (Volcano), this reduces the overhead of virtual function calls and materialization of intermediate results between operators, improves data locality, and enables further specialization with the context of multiple operators.

The same code generation framework is also used to speed up serialization/deserialization. As a unified data processing framework, Spark SQL supports flexible use of UDFs and type-safe Dataset APIs, both of which involve conversions between domain objects and Spark SQL internal data representations.

Data Sources

Spark separates computation and storage. The data sources are autonomous and can be shared with the other processing engines. The data schema is dynamic. The schema is just a virtual view that can be predefined or derived when reading it. For example, the built-in file source

connectors, including JSON, CSV, Parquet, and ORC, offer read-time schema inference. All the inferred or predefined schemas can be stored in a global persistent catalog, which is Hive metastore by default. The Hive metastore can also be shared with the other engines (e.g., Hive).

Thanks to the rich interoperability with external data sources, Spark SQL can read from, integrate with, and write to a variety of data sources. Users can use the built-in connectors and also plugin other third-party connectors. Through the data source APIs, third parties can build customized connectors to access the data stores.

Highly efficient vectorized readers are provided for columnar file sources (e.g., Parquet and ORC). Such complicated I/O operations are not fused into the whole-stage codegen. Bulk reading and processing can reduce the per-tuple interpretation overhead and leverage compiler optimization. The built-in cache mechanism is also columnar. The external sources and intermediate results can be explicitly cached in memory or local disk for reuse.

Conclusion

Since the initial release, Apache Spark has quickly become the largest open-source community in big data. It is the work of over 1000 contributors from over 250 companies. Spark SQL is the most active component in Apache Spark. Spark SQL brings end-to-end optimization in the sophisticated applications, which use one or multiple Spark libraries (e.g., SQL, MLlib, and structured streaming). More breakthroughs are expected in the coming years, as Spark SQL is growing to be a compiler of the unified analytics engine.

References

- Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'15)

- Dave A, Jindal A, Li LE, Xin R, Gonzalez J, Zaharia M (2016) Graphframes: an integrated API for mixing graph and relational queries. In: Proceedings of the 4th international workshop on graph data management experiences and systems (GRADES'16)
- Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A (2016) MLlib: machine learning in apache spark. *J Mach Learn Res* 17(1):34:1–34:7
- Michael A, Tathagata D, Joseph T, Burak Y, Shixiong Z, Reynold X, Ali G, Ion S, and Matei Z (2018) Structured Streaming: A declarative API for real-time applications in apache spark. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'18). 601–613
- Ousterhout K, Canel C, Ratnasamy S, Shenker S (2017) Monotasks: architecting for performance clarity in data analytics frameworks. In: Proceedings of the 26th ACM symposium on operating system principles
- Xin RS, Rosen J, Zaharia M, Franklin MJ, Shenker S, Stoica I (2013) Shark: SQL and rich analytics at scale. In: Proceedings of the ACM SIGMOD workshop on the web and databases (SIGMOD'13)
- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX symposium on networked systems design & implementation (NSDI'12)

Spark-Based RDF Query Processors

► Framework-Based Scale-Out RDF Systems

SparkBench

John Poelman and Emily May Curtin
IBM, New York, USA

Synonyms

[Apache Spark benchmarking](#); [Spark-Bench](#); [CODAIT/spark-bench](#)

Overview

SparkBench is a flexible framework for benchmarking, simulating, comparing, and testing ver-

sions of [Apache Spark](#) and Spark applications. It provides users three levels of parallelism and a variety of built-in data generators and workloads that allow users to finely tune their setup and get the benchmarking results they need.

Definitions

A framework for benchmarking Apache Spark.

Historical Background

Apache Spark began in 2010 as a research project by Matei Zaharia and others in the Berkeley AMPLab. Following the landmark success of *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing* by Zaharia et al. (2012), Spark continued to gain popularity and usage as its performance gains over traditional MapReduce workflows became evident. Spark continued to grow as well, introducing Python and R APIs, machine learning, graph computation, SQL, and streaming computation.

In 2015, a group of researchers at IBM identified a need in the growing Spark ecosystem for a benchmarking solution that focused specifically on Spark. While benchmarking systems such as the [AMPLab Big Data Benchmark](#) and Intel's [HiBench](#), to name just two, focused primarily on comparisons of Spark to other big data compute engines, the researchers at IBM chose to focus more specifically on Spark. In their 2015 paper *SPARKBENCH: A Comprehensive Benchmarking Suite For In Memory Data Analytic Platform Spark*, they wrote,

While Spark has been evolving rapidly, the community lacks a comprehensive benchmarking suite specifically tailored for Spark. The purpose of such a suite is to help users to understand the tradeoff between different system designs, guide the configuration optimization and cluster provisioning for Spark deployments. Existing big data benchmarks... are either designed for other frameworks such as Hadoop or Hive, or are too general to provide enough insights on Spark workload characteristics. (Li et al. 2015)

SparkBench Versus Other Tools

Intel's HiBench and other similar tools benchmark select Spark algorithms against implementations in other frameworks. These tools focus on a small selection of algorithms that are run in the style of traditional benchmarks. They are not focused specifically on Apache Spark, nor do they claim to be. Instead, they are useful for differentiating Spark against frameworks such as MapReduce, Storm, Hive, and others. SparkBench was designed to exercise various subsystems of Spark in a detailed way, not to compare Spark against other frameworks.

There are a number of projects that provide detail profiling information on Spark performance in a particular job. These, in most cases, can be combined with SparkBench to great effect by using SparkBench to build the simulation or benchmark in question and also drive the profiler. Some projects in this category include [CODAIT/spark-tracing](https://github.com/CODAIT/spark-tracing) (<https://github.com/CODAIT/spark-tracing>) and the commercially available [YourKit](http://spark.apache.org/developer-tools.html#profiling) (<http://spark.apache.org/developer-tools.html#profiling>).

Many production users of Spark have also observed the need for a production-grade Spark job scheduling tool. Some examples include Airflow by Airbnb and Azkaban by LinkedIn. While SparkBench provides the ability for users to compose suites of jobs to simulate use cases, it is not a production tool. [Airflow](#) and [Azkaban](#) provide feature-filled DAG schedulers and production monitoring for Spark jobs. They may be used as drivers for SparkBench to further enable complex simulation of real-world Spark use cases.

SparkBench Version 1.x Structure

The first version of SparkBench included a suite of individually compiled Spark applications. Each contained an application for generating data and an application containing the implementation of the algorithm in question. These applications were largely written in a mix of Java and Python and stitched together through a plethora of bash scripts that provided configuration and coalescing of results. The applications were designed to be individually compiled and run one at a time on a given cluster.

Key to the design of SparkBench 1.0 was the breadth of applications. In 2015, some of the now more widely used components of Spark including Spark SQL and Streaming were only just debuting. The authors of SparkBench 1.0 were careful to include representative workloads for each major piece of Apache Spark.

The source code for the first version of SparkBench was released on Bitbucket. It was eventually ported to GitHub under the umbrella of the CODAIT organization where contributors continued to update the code for subsequent releases of Apache Spark.

Motivation for Rewrite

Since the initial release of SparkBench in 2015, the adoption and number of use cases for Spark have continued to grow at a rapid pace. Notebooks now provide data scientists and analysts new ways of collaborating; cloud services provide businesses new ways of leveraging and accessing the cluster compute power of Spark; and Apache Spark itself has gone from version 1.4.0 to version 2.x in just 2 short years.

In 2017, a team at IBM working on performance improvements to core components of Apache Spark realized the need for a benchmarking tool that could do more than benchmarking. A simple run of traditional algorithms could not capture the complex nature of multiuser user cases that were the focus of the new improvements. To that end, Scala developer Emily May Curtin embarked on a major rewrite of SparkBench.

S

Project Structure

The new version of SparkBench (2.x) is a complete, ground-up rewrite of version 1.x. While version 1.x was a series of individually compiled Java or Scala programs with their own main() functions, the new version is a single Scala framework compiled using one SBT build file. Rather than algorithm implementations living alone in their own jar stitched together through bash, each is an implementation of the abstract workload class. All configurations are done in a single configuration file rather than a series of scattered variables in various bash scripts.

	Old version	New version
Project design	Series of shell scripts calling individually built jars	Scala project built using SBT
Build/release	Built manually, released manually	SBT, Travis CI, auto-release to GitHub releases on PR merge
Configuration	Scattered in shell script variables	Centralized in one configuration file with many new capabilities
Parallelism	None	Three different levels
Custom workloads	Requires writing a new subproject with all accompanying bash	Bring your own jar, plug it into existing framework with all configuration included

The new version of SparkBench features three independent levels of parallelism to enable users to dial in accurate settings for their particular simulation.

Foundations

SparkBench provides three levels of parallelism in which users can compose workloads and data generators to accomplish their benchmarking goals quickly and easily.

Workloads, Spark configuration, and other experiment structure are defined in one single config file written in [HOCON](#), a superset of JSON. They have a nested structure that starts with spark-submit configs and then goes down into workload suites and finally workloads at the bottom of the nested structure.

Workloads

The atomic unit of organization in SparkBench is the workload. Workloads are stand-alone Spark jobs that read their input data, if any, from disk, and write their output, if the user wants it, out to disk.

Some workloads are designed to exercise a particular algorithm implementation or a particu-

lar method. Others are designed to simulate Spark use cases such as multiple notebook users hitting a single Spark cluster.

Some existing categories of workloads include:

- ML workloads: Logistic Regression, K-Means, etc.
- “Exercise” workloads: designed to examine one particular portion of the Spark pipeline. A good example is SparkPi, a compute-heavy workload with very little disk IO.
- Data generators: SparkBench has the capability to generate data according to many different configurable generators. Generated data can be written to any storage addressable by Spark, including local files, HDFS, S3, etc.

Workloads can be composed with one another. They can be launched serially or in parallel.

Workload Suites

Workload suites are logical groups of workloads. Workload suites can be composed with each other for benchmarking tasks or to simulate different cluster use cases.

Workload suites control the benchmark results output and repetition of workloads. They can be composed with each other in serial or parallel fashion. The level of parallelism of the workload suite does not affect the level of parallelism for the workloads contained within each suite.

Spark-Submit Configuration

Under the hood, SparkBench converts users' configuration files into a series of spark-submit scripts. The spark-submit-config section of the configuration file allows users to change the parameters of those spark-submits.

Spark-submits, like other parts of the SparkBench configuration, can be composed serially or in parallel in order to accurately simulate different use cases. Multiple spark-submits additionally allow users to test different versions of Spark, different clusters, different configuration settings on the same cluster, and much more.

As with workloads and workload suites, spark-submits can be composed with each other and can be launched serially or in parallel. These three levels of parallelism (spark-submits, workload suites, and workloads) allow users fine-grained control over the setup of their simulation, so they can dial in the conditions to be as true to life as possible.

Example Configuration File

This is a configuration file used to benchmark performance of SQL queries against the same dataset stored in Parquet and CSV.

The first workload suite generates the data in CSV format, then picks up the CSV it just generated and rewrites it to Parquet format.

The second workload suite runs a total of four different workloads using the cross product of the two parameter lists. These four workloads are repeated ten times.

```
spark-bench = {
    spark-submit-config = [
        spark-home = "XXXXXXX" // PATH TO
        YOUR SPARK INSTALLATION
        spark-args = {
            master = "XXXXXXX" // FILL IN
            YOUR MASTER HERE
            executor-memory = "XXXXXXX" //
            FILL IN YOUR EXECUTOR MEMORY
        }
        conf = {
            // Any configuration you need
            // for your setup goes here,
            // like:
            // "spark.dynamicAllocation.
            enabled" = "false"
        }
        suites-parallel = false
        workload-suites = [
            {
                descr = "Generate a dataset,
                then take that same dataset
                and write it out to
                Parquet format"
                benchmark-output = "hdfs:///tmp/csv-vs-parquet/results-
                data-gen.csv"
                // We need to generate the
                // dataset first through the data
                // generator, then we take that
                // dataset and convert it to
                // Parquet.
                parallel = false
                workloads = [

```

```
                {
                    name = "data-generation-
                    kmeans"
                    rows = 10000000
                    cols = 24
                    output = "hdfs:///tmp/csv-
                    vs-parquet/kmeans-
                    data.csv"
                },
                {
                    name = "sql"
                    query = "select * from
                    input"
                    input = "hdfs:///tmp/csv-
                    vs-parquet/kmeans-
                    data.csv"
                    output = "hdfs:///tmp/csv-
                    vs-parquet/kmeans-
                    data.parquet"
                }
            ]
        },
        descr = "Run two different SQL
        queries over the dataset in
        two different formats"
        benchmark-output = "hdfs:///tmp/csv-vs-parquet/results-
        sql.csv"
        parallel = false
        repeat = 10
        workloads = [
            {
                name = "sql"
                input = ["hdfs:///tmp/csv-
                vs-parquet/kmeans-data.
                csv", "hdfs:///tmp/csv-vs-
                parquet/kmeans-data.
                parquet"]
                query = ["select * from
                input", "select '0', '22'
                from input where '0' <
                -0.9"]
                cache = false
            }
        ]
    }
}
```

S

Key Applications

SparkBench is in active use by developers working on contributions to Spark core. These developers use the project to compare the runtimes of

standard algorithms in the latest official version of Spark against their branch with changes.

Other developers have used the custom workload plugin capability of SparkBench to create custom workloads that exercise a very specific subsystem of Spark such as the caching mechanisms in core as they relate to dynamic allocation.

SparkBench additionally provides mechanisms for developers to simulate notebook environments such as [Jupyter](#) or [Apache Zeppelin](#). By inserting a Sleep workload between workloads, SparkBench users can build up a simulation of a notebook user. It is a simple extension to copy this workload suite to simulate multiple users.

This notebook simulation technique has been used to great effect by developers in the IBM Spark Technology Center who used SparkBench to debug and benchmark changes to Spark core involving resource contention settings.

Future Work

SparkBench is under active development by IBM and contributors from all over the world. Planned future work centers around several efforts.

1. *Launching through interfaces other than spark-submit (Livy, other REST services).* The structure of SparkBench should reflect the myriad of ways that users can interface with Spark. Adding REST capability to SparkBench allows for a new variety of use cases to be covered such as Spark as a service setups and other remote architectures.
2. *Expanding selection of built-in workloads.* At the time of publication, IBM developers are working on porting many workloads from version 1.x to 2.x.
3. *Allowing tighter integration with pluggable listeners and profilers.* SparkBench sits at the level of the application. To get more detailed profiling data on specific functions and do system profiling, users can employ a variety of profilers and tracers. In the future the output of

SparkBench should be synced and packaged with the output of tracers.

4. *Increase the number of standard big data benchmarks available by default, particularly TPC-DS.* The TPC-DS dataset and associated 100 SQL queries are a key standard benchmark for SQL systems. At the time of publication, developers at IBM are in progress adding workloads for TPC-DS data generation as well as the standard queries.

Cross-References

- [Apache Spark](#)
- [SparkBench](#)
- [TPC-DS](#)

References

- AMPLab Big Data Benchmark. <https://amplab.cs.berkeley.edu/benchmark>. Accessed 23 Feb 2018
- Apache Airflow. <http://airbnb.io/projects/airflow/>. Accessed 23 Feb 2018
- Apache Spark. <https://spark.apache.org/>. Accessed 23 Feb 2018
- Apache Zeppelin. <https://zeppelin.apache.org/>. Accessed 23 Feb 2018
- Azkaban. <https://azkaban.github.io/>. Accessed 23 Feb 2018
- HOCON (Human-Optimized Config Object Notation). <https://github.com/lightbend/config/blob/master/HOCON.md>. Accessed 23 Feb 2018
- IBM Spark-Tacing. <https://github.com/CODAI/spark-tracing>. Accessed 23 Feb 2018
- Intel HiBench Suite. <https://github.com/intel-hadoop/HiBench>. Accessed 23 Feb 2018
- Li M et al (2015) SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark. https://research.spec.org/fileadmin/user_upload/documents/wg_bd/BD-20150401-spark_benchmark-v1.3-spec.pdf. Accessed 23 Feb 2018
- Project Jupyter. <http://jupyter.org/>. Accessed 23 Feb 2018
- TPC Decision Support Benchmark. <http://www.tpc.org/tpcds/default.asp>. Accessed 23 Feb 2018
- YourKit Java Profiler. <https://www.yourkit.com/java/profiler/features/>. Accessed 23 Feb 2018
- Zaharia M et al (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf. Accessed 23 Feb 2018

Spark-Bench

- ▶ [SparkBench](#)

SPARQL Federation

- ▶ [Federated RDF Query Processing](#)

Spatial and Textual Data

- ▶ [Spatio-textual Data](#)

Spatial Data

- ▶ [Spatio-social Data](#)
- ▶ [Spatial Data Mining](#)

Spatial Data Integration

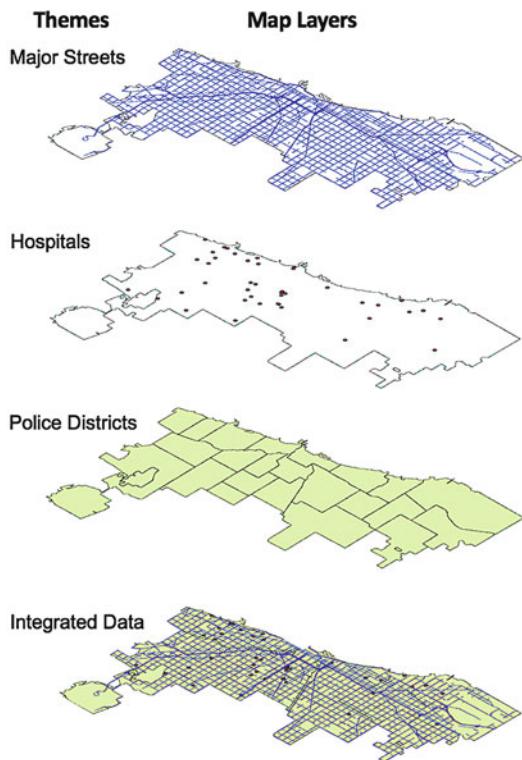
Booma Sowkarthiga Balasubramani and
Isabel F. Cruz
University of Illinois at Chicago, Chicago,
IL, USA

Synonyms

[Geospatial data integration](#); [Spatiotemporal Data Integration](#)

Definitions

Spatial data integration is a process in which different geospatial datasets, which may or may not have different spatial coverages, are made compatible with one another (Flowerdew 1991). The goal of spatial data integration is to facilitate the analysis, reasoning, querying, or visualization of the integrated spatial data. Figure 1 illustrates the integration of three layers or *themes*: major



Spatial Data Integration, Fig. 1 Spatial data integration (Chi 2017)

streets, hospitals, and police districts of the City of Chicago (Chi 2017).

Spatial data (often used synonymously with *geospatial data* and *geographical data*) refers to the information – location, shape, and relationships among geographic features – that describes the distribution of things on the surface of the Earth (DeMers 2008; Walker 1993).

Geospatial datasets that relate to physical geography, such as land and ocean boundaries, topography, weather, hydrology, natural disasters, land cover, and to human geography, such as administrative boundaries, land use, population, crime, buildings, water pipes, roads, points of interest, and many more, are freely available from a wide variety of sources (e.g., Socrata, data.gov). Many of those datasets vary both with respect to time and space. While each dataset may be important in its own right, it is often

their joint analysis that makes them critical for decision-making. Examples include finding the lakes in Maine (Egenhofer 2002), returning the towns affected by fires (Kyzirakos et al. 2014), or estimating the number of people affected by a water main break (Balasubramani et al. 2017).

Overview

Data are generated by various sources and different stakeholders, and they correspond to different locations and times. This section discusses such characteristics of spatial data that influence the techniques for spatial data integration.

Spatial and Spatiotemporal Data Types

Spatial data can be associated with (Flowerdew 1991):

Points: Sample points, selected randomly or for convenience (points in a survey, rain gauges), or points representing real objects (trees, buildings, cities).

Lines: Sample lines (paths for ecological sampling) or real phenomena (streets, rivers, geological faults).

Areas: With natural boundaries (islands, marshes) or artificial boundaries (land parcel, census tract).

Surfaces: A phenomenon defined across a region but measured only at discrete points (rainfall). Values at other points can be estimated (e.g., by interpolation).

For spatiotemporal data, we consider a single data type (Zheng 2015; Yoon and Shahabi 2008):

Trajectories: A trajectory of a moving object is a sequence of spatial points sampled at discrete instances of time. It is represented by a series of chronological points, $(p_1, t_1), (p_2, t_2), \dots, (p_n, t_n)$, where p_i is a two- or three-dimensional vector corresponding to the geospatial position observed at a time stamp t_i , where $i = 1, \dots, n$.

Spatial Data Formats

Spatial data are represented using one of the two data formats (Maffini 1987):

Vector: A coordinate-based data format that represents spatial data in the form of points, lines, and polygons. This format allows for the efficient encoding and operations to determine, for example, the eight basic topological relationships (Egenhofer 1993), namely, *disjoint*, *meet*, *overlap*, *contains*, *inside*, *covers*, *covered by*, and *equal*.

Raster: A data format that is composed of cells distributed in a regular grid, each containing an attribute value and location coordinates (Bishr 1998). This format encodes continuous data efficiently, such as in a topographic map that represents elevation using a color scale.

Conflation is the process of combining two or more sources representing the same geographic information so as to retain accurate data, minimize redundancy, and reconcile data conflicts (Longley 2005). The techniques for conflation vary based on the spatial data formats (Saalfeld 1988; Cobb et al. 1998). For example, vector-to-vector conflation involves matching vector data based on the similarities of the shapes of the vectors, geometric information, or the vector attributes. Vector-to-raster conflation exploits auxiliary spatial information (e.g., coordinates from a satellite image) and nonspatial information (e.g., the image resolution or color) and performs image processing to determine the correspondence among the datasets. Raster-to-raster conflation involves data-specific image processing techniques to extract and match various features (e.g., edges) across images (Dare and Dowman 2000).

Spatial Transformations

Spatial transformations are functions that obtain new spatial data from existing data, for example (Longley et al. 2015):

Buffering: Produces a new geometry by adding an area around the original objects (points, lines, areas).

Point-in-Polygon: Determines whether a given point lies inside or outside a given polygon.

Polygon Overlay: Outputs the polygons resulting from the intersections of two polygons.

Spatial Interpolation: Uses information from sampled points to guess the values at other points.

Density Estimation: Constructs an estimate of an unknown probability density function, based on observed data.

Heterogeneity

Differences in spatial data types and formats lead to two forms of heterogeneity that need to be resolved when integrating spatial data. Other obstacles for integrating data stem from differences in syntax, schema, or semantics (Bishr 1998). In addition, different data models, update frequency, resolution, and scale are also to be considered.

Spatial data exhibit various kinds of heterogeneity (Beck et al. 2008; Hakimpour 2003; Bishr 1998):

Syntactic Heterogeneity: Spatial data is stored in different formats (e.g., shapefile, KML). The data format dictates the tools and approaches to query, integrate, and analyze the data.

Structural or Schematic Heterogeneity: This kind of heterogeneity arises when the data model is the same (e.g., relational) but the schemas differ, including the type of an attribute (for example, °F vs. °C). Further heterogeneities occur when the same concept is in one of the table names in a schema and is a name of an attribute in another schema.

Semantic Heterogeneity: Semantic heterogeneities occur when the same entity has different names, or when different entities have the same name. For example, *Chicago* and *Windy City* represent the same entity, whereas *Square* represents a geometry with four sides of equal length or a *City Square*.

Spatial Modeling Heterogeneity: The same geographical phenomena may be modeled differently, for example, *street* as a *line* in one dataset, and as a *polygon* in another dataset.

Resolution or Granularity: There are three kinds of resolution: *spatial*, *temporal*, or *semantic*. *Spatial resolution* is the grain size or the cell size of spatial data, and represents the amount of detail that can be observed in the spatial dataset (Longley 2005). For instance, satellite imagery with a resolution of 2.4 m is more detailed than a satellite imagery with a resolution of 1.1 km. *Temporal resolution* is the time interval during which the value remains the same, for example, temperature that is published every hour, but the estimated time of arrival for a flight may be updated every five minutes. *Semantic resolution* refers to the level of specification of a real-world entity (Fonseca et al. 2002). For example, a body of water may be perceived as a lake or as a water body, depending on the application. Semantic similarity can be measured by the distance that separates two concepts in a classification hierarchy (e.g., an ontology).

Scale: Often used synonymously with resolution, scale describes the dimensions of spatial phenomena and determines its object type (Longley 2005). The details of the spatial phenomena represented in the spatial dataset heavily depends on scale. For example, the New York city is represented as a one-dimensional point on the world map, but is represented as a two-dimensional area on the city map, which is more detailed.

S

Key Research Findings

Designing spatial data integration systems is characterized by various challenges that spatial data exhibit. Quite justifiably, significant research effort has been invested over the past decades in the spatial data integration problem. This section discusses some of the most important works that focus on integrating spatial data in a seamless manner.

Spatial Entity Resolution

Entity resolution in spatial data integration refers to the process that determines whether two locations referenced by two or more different data

sources correspond to the same real-world entity (Sehgal et al. 2006). Some common approaches for entity resolution in spatial data integration are as follows:

One-Sided Nearest-Neighbor Join: This method associates data with location a in dataset A to data with location b in dataset B , if b is the closest location to a among all other locations in B (Batini and Scannapieco 2016).

Image Processing Techniques: Integration of digital maps (Doytsher et al. 2001; Saalfeld 1988) involves image processing techniques and requires complete information about the spatial entity. Satellite imagery has long been used to integrate maps.

Feature-Based Matching: These algorithms use location features, that is, attributes such as spatial coordinates, location name, and location type independently, and find mappings based on a similarity metric such as *edit distance* or *coordinate similarity*.

Feature-Based Integration: This method combines different similarity measures. For example, a feature vector consisting of the similarity metrics for spatial and nonspatial features (e.g., *edit distance* and *coordinate similarity*) is associated with each pair of locations. Final similarity is determined using is performed using machine learning techniques like logistic regression or neural networks.

Uncertainty

Uncertainty, which is the difference between an occurrence in reality and its representation in a database (Zhang et al. 2014), is omnipresent in geospatial data. Uncertainty occurs in several phases ranging from geographical abstraction, data acquisition, limited observations, different but equally valid interpretations of the geography, geo-processing techniques, and representation of the results of geographic computations to the use of the results (Devillers and Jeansoulin 2006; Zhang and Goodchild 2002). When datasets are integrated, the resulting uncertainty stems from the individual datasets as well as from the integration process itself.

Spatial Data Integration: Observations

This section highlights some details observed during spatial data integration (Goodchild et al. 1992; Cruz and Xiao 2008; Mohammadi et al. 2006; Cruz et al. 2007):

1. The set of functions that can be applied or the type of analysis that can be performed on spatial data heavily depends on the data models and probability distribution functions. For example, interpolation can be performed on point samples of mineral deposits to determine the distribution of mineral reserves for an area. However, one cannot interpolate the point samples representing occupancy in high-rise buildings to estimate the population of a city.
2. Data comes from various sources such as social media, private and public organizations, and government agencies. Each of these sources handles data using different database management techniques, formats, and scales. Even in the same organization, for example, in a city, the boundaries of the various districts (school, asphalting, sanitation, forestry) may not coincide, thus complicating correlations across different datasets.
3. The same data can be heterogeneously represented depending on the data provider, either because of the data collection method, storage and management techniques, or, sometimes, the functions used to derive the data.
4. The availability of metadata does not guarantee in itself that data will be correctly integrated. The semantics of the different datasets, the target model in the case of cross-domain data integration, the goal of the integration, and other factors are important.
5. Attribute-level metadata, which contains the data accuracy, date collected/updated, or source of a particular attribute, is usually more useful than the generic theme-based metadata, which focuses on the domain of the spatial data (e.g., agriculture, environmental) and consists of information including the spatial extent, content type, and publisher.
6. Sources with different privacy requirements further add to the data heterogeneity and to

the complexity of the data integration process. For example, data may have to be previously transformed to a coarser detail or integrated using privacy-preserving techniques. Another known risk stems from the fact that the integrated datasets may allow for a more detailed analysis than was previously possible for each dataset.

State-of-the-Art Techniques for Spatial Data Integration

Due to the complex nature of spatial data, a data integration framework cannot be generalized to all spatial data integration cases. Geostatistical techniques are specific to spatial data (Gotway and Young 2002), while others that use ontologies can be used for the integration of spatial (Fonseca et al. 2002) and nonspatial datasets (Cruz and Xiao 2005). Ontologies are knowledge representation structures that define concepts and their relationships explicitly.

Geostatistical Techniques

Geostatistical techniques for data integration aim to allow for the synthesis of data at different scales. One of the problems is the fact that spatial data is organized, subdivided, and stored based on spatial units (e.g., land parcel, grid cell of size 1 km²), which may differ across datasets.

Spatial data transformations are special cases of the *change of support problem* (COSP), where *support* includes the geometrical size and shape of a region associated with a measurement (Gotway and Young 2002). Examples of COSPs include block kriging and areal interpolation. For example, block kriging applies to the case where point values are observed but the nature of the process is associated with an area, whereas areal interpolation applies to the case where area values are observed for a set of source polygons (e.g., school districts) but need to be converted to a set of target polygons (e.g., census tracts).

Ontology-Based Approaches

To integrate datasets that exhibit different spatial (or temporal) resolutions and are different in terms of schema (structure that represents the organization of data) and semantics, an ontology

can act as a mediator between the two datasets so as to establish relationships among the attributes of the datasets (Tran et al. 2015). For example, an urban spatial ontology can be used to establish relationships such as Chicago is *inside* Illinois. It is also possible to use an ontology for each dataset and establish correspondences between the ontologies, using rules defined by the domain experts (Mena et al. 2000). Also, there are different types of ontologies with respect to their expressiveness.

A set of mappings, which defines the correspondences among the concepts in the ontologies, are established to describe topological relations (such as *disjoint*, *meet*, or *overlap*). For resources with rich quantitative spatial information, the focus is on obtaining mappings between spatially colocated regions. The similarity between two geometric shapes is then obtained using the Hausdorff distance, which is defined as the maximum distance of a set to the nearest point in the other set (Rote 1991).

However, two concepts may be matched even if there are other pairs that are better matched. Structural methods (Cruz and Sunna 2008; Melnik et al. 2002) consider the structure around each of the two concepts being matched, that is, the concepts that are adjacent to them. Then, the two concepts match if their adjacent concepts also match.

Examples of Application

Apart from facilitating complex geospatial data analytics, spatial data integration also enables data accessibility (i.e., data storage and retrieval) and data interoperability (i.e., compatibility with other datasets). Some applications that benefit from spatial data integration are described next:

Utility Data Integration: This application facilitates the exploration and analysis of the impact of disruptions in urban infrastructure systems. There are several frameworks that address the challenges in integrating utility data from different perspectives (Beck et al. 2007, 2008; Psyllidis et al. 2015).

Land-use Matching: This application maps two different land-use ontologies (Cruz and Xiao 2008; Cruz et al. 2007), which classify land parcels based on their usage, for example, residential, industrial, agricultural, or commercial, so that a single query can retrieve those parcels across different classifications.

Spatial Phenomena Correlation: This application computes the correlations between different spatial phenomena such as crop rotations and biodiversity (Tran et al. 2015) by integrating a land-use dataset and a biology dataset.

Environmental and Disaster Analysis: This application uses the query language SPARQL extended with geometric objects, set operations on those objects (e.g., union), and the result of spatial transformations on those objects (e.g., buffer), to ask queries that return the names of the towns that have been affected by fire and to construct new geometric objects, for example, the burnt areas in coniferous forests (Kyzirakos et al. 2014).

Future Directions for Research

Each of the challenges associated with big data – volume, variety, variability, velocity, and value – is patent in geospatial data integration. A comprehensive environment for integrating and analyzing geospatial data has several layers, each being conceptually complex when considered on its own and even more so when considering the interactions among them. Those layers include data extraction, data translation, ontology extraction, syntactic and semantic matching, spatiotemporal matching, application, visualization, and analytics, together with two crosscutting layers: machine learning and querying (Cruz et al. 2013).

Ontology-based approaches resolve structural and semantic heterogeneities. These are expressive techniques that enable cross-database queries and reasoning on the spatial, temporal, as well as the spatiotemporal relationships. However, ontologies are often not available for the domain of interest and even when available can be location

dependent. For example, an ontology of crime in Chicago may not apply to New York City. Land-use ontologies also vary across municipalities and counties (Wiegand et al. 2002) but can be created from the corresponding taxonomies. It is, however, nontrivial to incorporate the associated unstructured information in the ontology. Hence, the creation, reuse, and evolution of ontologies need to be streamlined.

In addition to the expressiveness issues, other issues pertain to the computational efficiency, for example, of record linkage (Felleji and Sunter 1969; Kelley 1984), large ontology matching (Faria et al. 2017), or geostatistical scale mapping (Gotway and Young 2002). When several of these strategies need to be applied, their optimization is necessary, possibly along with the determination of the order of execution of the various layers to improve overall efficiency.

Cross-References

- ▶ [Data Integration](#)
- ▶ [Holistic Schema Matching](#)
- ▶ [Probabilistic Data Integration](#)
- ▶ [Record Linkage](#)
- ▶ [Schema Mapping](#)
- ▶ [Uncertain Schema Matching](#)

References

- (2017) Open data portal of the city of Chicago. <https://data.cityofchicago.org/>. Accessed: 30 Nov 2017
- Balasubramani BS, Belingheri O, Boria ES, Cruz IF, Derrible S, Siciliano MD (2017) GUIDES—geospatial Urban infrastructure data engineering solutions. In: ACM SIGSPATIAL international conference on advances in geographic information systems
- Batini C, Scannapieco M (2016) Data and information quality: dimensions, principles and techniques. Springer, Cham
- Beck AR, Fu G, Cohn AG, Bennett B, Stell JG (2007) A framework for utility data integration in the UK. In: 26th urban data management symposium, Stuttgart, 10–12 Oct 2007. Taylor & Francis, pp 261–276
- Beck AR, Cohn AG, Sanderson M, Ramage S, Tagg C, Fu G, Bennett B, Stell JG (2008) UK utility data integration: overcoming schematic heterogeneity. In: International conference on advanced optical materials

- and devices. International Society for Optics and Photonics, 71431Z
- Bishr YA (1998) Overcoming the semantic and other barriers to GIS interoperability. *Int J Geogr Inf Sci* 12(4):229–314
- Cobb MA, Chung MJ, Foley III H, Petry FE, Shaw KB, Miller HV (1998) A rule-based approach for the conflation of attributed vector data. *GeoInformatica* 2(1):7–35
- Cruz IF, Sunna W (2008) Structural alignment methods with applications to Geospatial ontologies. *Trans GIS: Special Issue on Semant Similariy Meas Geospat Appl* 12(6):683–711
- Cruz IF, Xiao H (2005) The role of ontologies in data integration. *J Eng Intell Syst* 13(4):245–252
- Cruz IF, Xiao H (2008) Data integration for querying geospatial sources. Springer, Boston, pp 113–137
- Cruz IF, Sunna W, Makar N, Bathala S (2007) A visual tool for ontology alignment to enable geospatial interoperability. *J Vis Lang Comput* 18(3):230–254
- Cruz IF, Ganesh VR, Caletti C, Reddy P (2013) GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics. In: ACM SIGSPATIAL international symposium on advances in geographic information systems (ACM GIS). ACM, pp 544–547
- Dare P, Dowman I (2000) A new approach to automatic feature based registration of SAR and SPOT images. *Int Arch Photogramm Remote Sens* 33(B2):125–130
- DeMers MN (2008) Fundamentals of geographic information systems. Wiley, Hoboken
- Devillers R, Jeansoulin R (2006) Fundamentals of spatial data quality. ISTE Publishing Company, London/Newport Beach
- Doytsher Y, Filin S, Ezra E (2001) Transformation of datasets in a linear-based map conflation framework. *Surv Land Inf Syst* 61(3):165–176
- Egenhofer M (1993) A model for detailed binary topological relationships. *Geomatica* 47(3):261–273
- Egenhofer MJ (2002) Toward the semantic geospatial Web. In: ACM symposium on advances in geographic information systems. ACM GIS, pp 1–4
- Faria D, Pesquita C, Mott I, Martins C, Couto FM, Cruz IF (2017, in press) Tackling the challenges of matching biomedical ontologies. *J Biomed Semant* 9:1–19
- Fellegi IP, Sunter AB (1969) A theory for record linkage. *J Am Stat Assoc* 64(328):1183–1210
- Flowerdew R (1991) Spatial data integration. *Geogr Inf Syst* 1:375–387
- Fonseca F, Egenhofer M, Davis C, Câmara G (2002) Semantic granularity in ontology-driven geographic information systems. *Ann Math Artifi Intell* 36(1–2):121–151
- Goodchild M, Haining R, Wise S (1992) Integrating GIS and spatial data analysis: problems and possibilities. *Int J Geogr Inf Syst* 6(5):407–423
- Gotway CA, Young LJ (2002) Combining incompatible spatial data. *J Am Stat Assoc* 97(458):632–648
- Hakimpour F (2003) Using ontologies to resolve semantic heterogeneity for integrating spatial database schemata. Ph.D. thesis, Universität Zürich
- Kelley RP (1984) Blocking considerations for record linkage under conditions of uncertainty. Technical Report CENSUS/SRD/RR-84/19, U.S. Bureau of the Census, Statistical Research Division, Washington, DC, 709133
- Kyzirakos K, Karpathiotakis M, Garbis G, Nikolaou C, Bereta K, Papoutsis I, Herekakis T, Michail D, Koubarakis M, Kontoes C (2014) Wildfire monitoring using satellite images, ontologies and linked geospatial data. *J Web Sem* 24:18–26
- Longley P (2005) Geographic information systems and science. Wiley, Hoboken, pp 148–149
- Longley PA, Goodchild MF, Maguire DJ, Rhind DW (2015) Spatial data analysis, chap 13, 4th edn. Wiley, Hoboken
- Maffini G (1987) Raster versus vector data encoding and handling: a commentary. *Photogramm Eng Remote Sens* 53(10):1397–1398
- Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: IEEE international conference on data engineering (ICDE), pp 117–128
- Mena E, Illarramendi A, Kashyap V, Sheth AP (2000) OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distrib Parallel Databases* 8(2):223–271
- Mohammadi H, Binns A, Rajabifard A, Williamson IP et al (2006) Spatial data integration. In: 17th UNRCC-AP conference and 12th meeting of the PCGIAP, Bangkok, pp 18–22
- Psyllidis A, Bozzon A, Bocconi S, Bolivar CT (2015) A platform for Urban analytics and semantic data integration in city planning. In: International conference on computer-aided architectural design futures. Springer, pp 21–36
- Rote G (1991) Computing the minimum hausdorff distance between two point sets on a line under translation. *Inf Process Lett* 38(3):123–127
- Saalfeld A (1988) Conflation: automated map compilation. *Int J Geogr Inf Syst* 2(3):217–228
- Sehgal V, Getoor L, Viechnicki PD (2006) Entity resolution in geospatial data integration. In: ACM international symposium on advances in geographic information systems. ACM GIS, pp 83–90
- Tran BH, Plumejeaud-Perreau C, Bouju A, Bretagnolle V (2015) A semantic mediator for handling heterogeneity of spatio-temporal environment data. In: Metadata and semantics research. Springer, Cham, pp 381–392
- Walker R (1993) AGI standards committee GIS dictionary. Association for Geographic Information, London
- Wiegand N, Patterson D, Zhou N, Ventura S, Cruz IF (2002) Querying heterogeneous land use data: problems and potential. In: National conference for digital government research (dg.o), pp 115–121
- Yoon H, Shahabi C (2008) Robust time-referenced segmentation of moving object trajectories. In: IEEE

- international conference on data mining (ICDM). IEEE, pp 1121–1126
- Zhang J, Goodchild MF (2002) Uncertainty in geographical information. CRC Press, London
- Zhang J, Atkinson P, Goodchild MF (2014) Scale in spatial information and analysis. CRC Press, Boca Raton
- Zheng Y (2015) Trajectory data mining: an overview. ACM Trans Intell Syst Technol (TIST) 6(3):29

Spatial Data Mining

Shuliang Wang and Tisinee Surapunt
Beijing Institute of Technology, Beijing, China

Synonyms

[SDM theory](#); [Spatial data](#)

Definitions

The growth of spatial data which plays a part in the agricultural products, sustainable development, and human society development is accumulated continuously. Not only the size and volume are immense, the structure is also convoluted with the abundant and deep of their contents. The spatial dataset is full of the information and experience collection from geomatics that relates to Remote Sensing (RS), Global Positioning System (GPS) and Geographic Information System (GIS). A wide variety of databases consist of electronic maps and planning network from their infrastructure. With the increase in the spatial data collection, the processes of gathering, management, and transmission data require the powerful techniques. The traditional methods lag of the ability of big data query. Thus, the Spatial Data Mining (SDM) becomes the suitable technique. The Knowledge Discovery from Geographical Information System database (KDG) approach can support SDM to be more developed in data mining and geomatics (Li and Cheng 1994). The discovered knowledge from spatial datasets can support a decision-making system on various areas such as urban planning

and construction, transportation, resource allocation, capital optimization, marketing, and medical treatment. Then, SDM benefits the global sustainability. However, the large volume of spatial datasets will impact definitely the execution time. Then, the computerization which is composed of a computer chip, a power of a central processing unit, and a transfer rate of communication channel will boost up an acceleration of an interactive process. Presently, the artificial intelligence (AI) and machine learning (ML) are important roles instead of the manual process. AI is in charge with the human deterministic intelligence simulation by three strategies which are symbolism, connectionism, and behaviorism. While the ML supervises on the human learning simulation which obtains the knowledge form expert systems automatically, SDM will request the AI and ML to improve the data mining capability.

Recently SDM is popular to present the discovered knowledge through the real-world application but should concern over the crisp data or uncertain data. In that case, the probability theory, spatial statistics, rule induction, clustering analysis, spatial analysis, fuzzy set, data fields, rough sets, genetic algorithms, visualization decision trees can support those issues (Ester et al. 2000).

Overview

SDM Concepts

The SDM can enhance the human ability for extracting, applying, and transforming the spatial data in the real-world application. Because the ML and AI belong to the SDM procedure, the SDM is more understandable by both mathematic and non-mathematic theories. So, the SDM is a technique to clean, sample, and convert a vast and complex spatial data.

The SDM Pyramid is a symbol of SDM Concepts which concentrates on transforming from the spatial data to information and knowledge. The process starts with the data preparation, data mining, and post-processed of data mining. If the description is more abstract,

coherent, and general, the technologies will be more deep and advanced. Piatetsky-Shapiro (1994) reviewed the concept of data, information, and knowledge pyramid (DIKP). The study proposed that the different levels of data presented the different concept element. So, it was impossible to make the association between each concept. Han et al. (2012) demonstrated the visual process of data mining by different graphics. However, their work was imperfect to present the role and difference of elements. It implied that the complexity of spatial data was difficult to depict the levels of data. It is compulsory to integrate DIKP and the process of SDM in order to clarify the concept and roles of SDM. Thus, the SDM Pyramid in Fig. 1 is presented as a result. The SDM Pyramid specifies the level of spatial data from bottom to top. Each level of data presents the different element concept. The SDM pyramid is the main solution of transferring and managing data between layers. For example, the RS images, the images data are analysed and usually take the resolution in distinct layers. The top layer is the coarsest resolution and small scale, while the bottom layer is the finest resolution and largest scale. Then, the data is indexed and well-organized can browse in the multi-source, multi-scale, and cross-resolution image. The SDM pyramid manifests the data characteristics from spatial knowledge to the spatial data in real world. The amount of spatial data directs variation to the data complexity. The more simple data is, the more spatial data quantity contains.

Spatial Data to Spatial Knowledge

The SDM applies an interdisciplinary methodology to acquire spatial knowledge effectively. Each unit of spatial numerical values, spatial data, spatial information, and spatial knowledge presents the dissimilar concepts but connects in detail.

- Spatial Numerical

The spatial numerical represents between a specific value and a measurement unit. The spa-

tial numerical is a transmission when the spatial objects are gathered, transformed, and applied by the computerized SDM.

- Spatial Data

The spatial data can represent raw data or processed data. The processed data illustrates digital and nondigital data such as numbers, words, symbols, graphics, images, video, and language which utilizes the raw data. The spatial data can be a reference in order to know spatial objects and numerical nature with various attributes.

- Spatial Concept

The spatial objects together with their implication and extension describe the spatial concept. The spatial concept may involve in the problem-solving and indicate the states of spatial objects

- Spatial Information

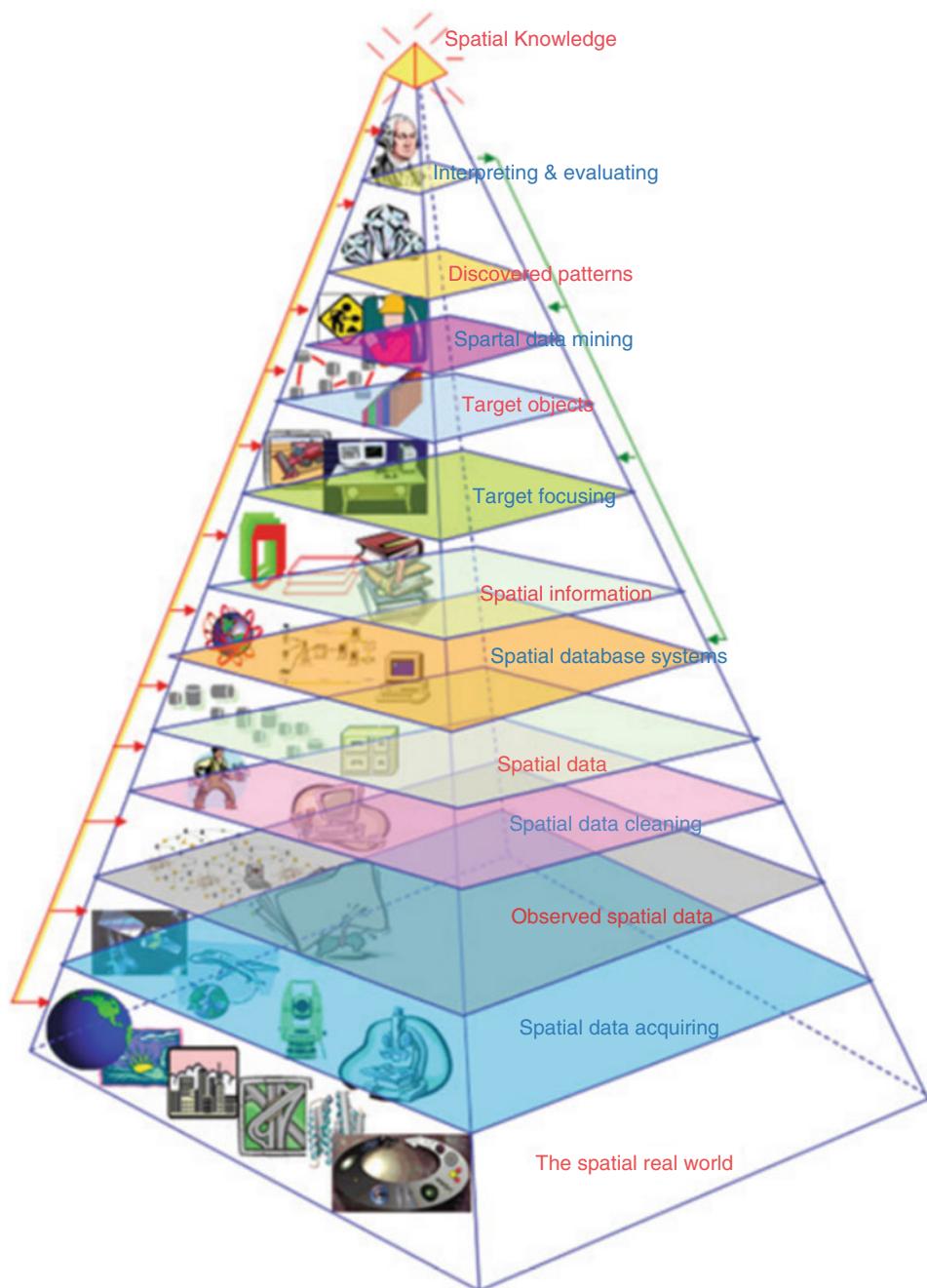
The analysis of spatial objects in the dataset can determine the spatial information. The spatial information demonstrates the meaningfulness and eliminates the possible uncertainty. It is imperative to be the decision-maker.

- Spatial Knowledge

The spatial knowledge is built from one or more parts of the spatial information. The spatial knowledge is depicted through the processes of correlation, association, classification, and clustering. Both spatial information and spatial knowledge require SDM in the data processing.

- Unified Action

The spatial data carries information to specify their properties, quantities, location, and relationship. So, the unified actions of spatial numerical values, spatial data, spatial concept, and spatial information become the spatial knowledge.



Spatial Data Mining, Fig. 1 SDM Pyramid (Li et al. 2015)

Spatial Knowledge to Discover

In an effort to discover the various spatial knowledge, SDM requires various techniques under “rule plus exception” with different views. The

miscellaneous preset rules have already qualified on the practical manipulation such as General Geometric Rule and Spatial Association Rule, Spatial Characteristics Rule and Discriminate Rule,

Spatial Clustering Rule, Classification Rule, Spatial Predictable Rule and Serial Rule and Spatial Exception or Outlier. Therefore, the spatial knowledge is extracted properly to be the Discover (Li and Du 2007).

Spatial Knowledge Representation

SDM promotes the spatial knowledge representation in order to demonstrate the knowledge. The integration of traditional expression methods can enhance the conversion of the knowledge expression method efficiently. When the knowledge extraction finishes, the knowledge measure can bound the level to depict the discovered pattern which is meaningful and interesting. A natural language, predicates logic, function model, characteristic table, semantic network, and Petri net are fundamental representations. For example, the natural language which is a general knowledge recognition of uncertainty can interpret spatial datasets based on human thinking. It is an effective tool to reorganize the thinking process. The concept is to map the object from the objective to the subjective cognition. Hence, the more the knowledge is abstract, the more the natural language is proper.

SDM Data Source

Contents and Characteristics of Spatial Data

The primitive data sources of spatial data are geographic contents. The spatial data acquisition gathers from radar, infrared, photoelectric, satellite, digital cameras and telescopes. There are three acquisition methodologies which are point acquisition, area acquisition, and mobility acquisition. The point acquisition is performed by the GPS receivers which collect the coordinates and attributes of surface points on Earth. The area acquisition targets the large areas of images which are geometrical and physical features. The mobility acquisition incorporates with GPS, RS, and GIS to observe the Earth system.

After gathering data, the main characteristics of spatial data which are size, complexity, dimension, and certainty definitely differentiate

from the common data. The location-based spatial and no-spatial data imply the implication of objects. The spatial object is a core of an object-oriented data model such as point, line, area, or complex objects in the large dataset. A point can be a single, directed, and grouped point which presents the location on the Earth surface without shape and size in the computerized system. Then the interconnection between points presents a line as rivers and roads. The line presents the spatial curve of Earth surface as linear from point-to-point or the reticulated pattern. After that, the shape and size of lines illustrate an area as the curved surface of Earth. Finally, a complex object combines more than two objects of point, line, and area.

To enhance the efficiency of the target, three main characteristics of spatial data are time, location, and theme. Due to dynamic changes, time is sensitive to collect and compute the spatial data. The gathered data should be affirmative and reliable such as weather forecast. Next, the location determines the location relationship. The topology and relationship are demonstrated by position, azimuth, shape, and size of a spatial object. Then, the theme is an extraordinary characteristic which defines the land endurance, pH value, land cover, and population density. Some developed technology such as RS is necessary to request more thematic data acquisition. The spatial data is compulsory to utilize carefully because their characteristics impact the occurrence of spatial data fusion easily. However, the spatial data uncertainty is ubiquitous during the implementation. To maintain the consistency and the integration of spatial data for bringing out spatial objects, all techniques are required to solve those issues.

Spatial Data Model

A common use of spatial data models are hierarchical model, network model, relational model, and object-oriented model. Both hierarchical and network models address the relationship between objects which consider the path of data, storage, and accessibility. The hierarchical model of 1 or more than 1 root node(s) and leaf node(s) can store and access the hierarchical database

efficiently although users' interaction are inconvenient. The network model relationship is the connection between arbitrary records with others. Then, the occurrence of multiple relationships is possible to depict as an undirected graph even if it is the irregular structure. The relational model composes of row as an entity record and column as an entity attribute. When the model concerns the relationship between two connected dimensional tables with conditions, the unique and foreign keywords require. The unique keyword is a single or composited attribute for indicating the entity, while the foreign keyword is the relationship between entities. If relational model contains the Boolean logic and arithmetic rules, the Structured Query Language (SQL) has to manage the data. The Object-Oriented model has been highly valued in GIS application since late 1980s and early 1990s. The spatial objects relationship is expressed by the generalization, union, and aggregation. After that the model was integrated between the object-oriented and database techniques. When the complicated relationship or nested relationship exists, it is better to present as a table structure.

Spatial Databases

The spatial database is built for being a tool equipment and data manipulation. The definition, representation, and storage of spatial data diverge the common transactional data. The spatial data can be collected by the data manipulation of surveying and mapping database. The different purposes will be fulfilled with the different database. There are general database, graphic database, and image database. In addition, the digital elevation model (DEM) database is productive data arrangement. When a unified spatial index is developed, users can find the data of any domain fast. The large-scale databases are also not against the performance of data display. The spatial index allows users to access data by retrieving and transferring among different levels effortlessly.

From GGDI to Big Data

The GGDI which stands for Global Geospatial Data Infrastructure can run on the Digital

Earth. GGDI to Big data is from 1970s which published the "Information Society." The study discussed on a global information infrastructure with the telecommunication infrastructure. After 10 years, the "Centralized Land Information Database" introduced a more complex network of distributed land information. The data resource management was adopted from a common approach to a computer based. Then in 1990s, the Spatial Data Infrastructure (SDI) was originated to support and accelerate the geographic data exchange standards. The local interests, demands, and constraints of country or organization are effected by the developed GGDI.

Presently, developers who usually apply the new technology for supporting their work, study and life, the digital contents are invented to present in form of internet of things and cloud computing. Thus, the big data seems to be an essential source for the digital content applications. The simulation of digital earth becomes popular in many countries. Most of the services utilize the technology of RS, GIS, ML, AL, and Internet of thing. The applications on network mapping, Web 2.0, and mobile sensors on vehicles impact significantly to the users' experience. Therefore, the spatial information system is obvious different from a traditional information system. The new geo-information systems are professional to serve the data and publish to all users. Hence, the new geo-information era improves the whole system of the geospatial information industry to prosperity. Soon the geo-information benefits for all aspects and leads to smart city definitely.

Key Research Findings

Bottleneck of SDM

Because of complex spatial entities, geographic location of features, boundaries, and relationship, SDM is capable to identify. However, the bottleneck of SDM is attentive. Some researches were

achieved with those goals but some difficulties need to be solved.

- Excessive Spatial Data

The geo-graphical information is imperative to accomplish the application development. The data source comes from RS, digital techniques, networks, multimedia, and images. So, the spatial database expands and grows rapidly. With the issues of immense dataset, the traditional implementation cannot manipulate with tons of spatial data such as the enumeration method of data analysis. Therefore, the data mining (DM) was introduced and capable to improve.

- High-Dimensional Spatial Data

The spatial data grows in both horizontal and vertical attributes, but the GIS information is still inadequate to illustrate the spatial data with high-dimension structure. The discovered knowledge can resolve the challenged. For example how to accelerate for searching and query the target data or how to decline the number of dimensional data.

- Polluted Spatial Data

The spatial entities may contain both useful and polluted data. The organized spatial data should concern about position accuracy, attribute accuracy, consistency, lineage, and integrity. The polluted spatial data is in a form of incompleteness, dynamic changes, noise, redundancy, and sparsity. They can cause a low-quality standard of data accuracy.

- Uncertain Spatial Data

The uncertain spatial data in SDM is impossible to avoid because of the approximate sample data and abstract mathematical models. Due to the crisp set, probability theory, GIS model, the sensitivity analysis for SDM can normalize to be a certain data. However, a related special entity of

every single attribute can be emphasized instead of spatial data uncertainty.

- Mining Differences

The SDM utilization succeeds in understanding and gathering the different cognitive hierarchy of spatial dataset. The different mining on different aspects will return the different results from the results of discovered knowledge despite the same database. In addition, the making use of SDM can present a hierarchical decision-maker's awareness among those differences.

- Problems to Represent the Discovered Knowledge

The natural language is the best way to illustrate the conceptual knowledge which is a kind of the discovered knowledge. An exclusion of data uncertainty in SDM process will return incomplete knowledge and be full of errors although the adequate techniques are provided.

Methods and Techniques in SDM

The variety of methods and techniques impact directly the discovered knowledge in SDM. Each method supports the different characteristics based on the context. The SDM also concerns the evolution of human thinking network and optimal solution processes.

S

Crisp Set Theory

The Crisp Set Theory was introduced by Cantor in nineteenth century based on the modern mathematics. The theory approaches on probability, evidence theory, spatial statistics, spatial analysis, and data field. The probability which was conducted by Arthurs in 1965 works properly in the stochastic probabilities with randomness spatial data. Actually, the probability is unlike from the likelihood. The big probability does not present the high likelihood and vice versa.

Then, the evidence theory is known as Dempster-Shafer theory or significance theory. When the evidence exists, the hypothesis

evaluates a minimum of the belief function. Hence, the hypothesis cannot be denied because the plausibility function measures the maximum degree. However, the evidence is null by the unsupported interval. The evidence theory can also be an identification to probability theory. It implies that the evidence theory is the extension of the probability theory. The evidence theory is divided into two parts by the SDM technique which are the certainty measures confidence and the uncertainty measures likelihood. So, the framework of evidence data mining (EDM) concentrates on mass functions and mass operators to discover the knowledge.

Moreover, the spatial clustering presents the concept of similarity. The cluster determines the dataset to maximize the similarity and minimize the dissimilarity between clusters. The data is clustered by the object characteristics through spatial raw data. Only the basic object attributes can figure directly the meaningful clusters in a spatial dataset. Then, the data point in a multi-dimensional feature space can also cluster for the pattern recognition.

Extended Set Theory

SDM is capable on the similarity of human thinking on certainty and uncertainty. The crisp set examines the certainty, while the extended crisp set considers the uncertainty. To classify which group elements belong to, the crisp set bounds the binary logic of 0 and 1 as the full membership or no membership. However, the indeterminate boundaries are not accurate with the crisp set methods. Hence, SDM enhances the crisp set theory on uncertainty by using fuzzy set, rough set, and cloud model.

The fuzzy set is the fuzziness on a fuzzy membership. A close interval as [0, 1] of the partial membership defines an uncertainty probability. Two main fuzzy techniques are the fuzzy comprehensive evaluation and fuzzy clustering analysis. The fuzzy set is more proper to classify the spatial heterogeneous distribution of geographical uncertainty. The more fuzziness is created, the more difficult and complex system is. The approximation of the elements is close to 1 means the more possible belong to the target class.

Next, a rough set focuses on an incompleteness of the lower and upper approximations in SDM. An incompleteness-based reasoning method fulfills a decision-making by the characteristic of certainty and uncertainty data. The rough set favors on the partial membership with a set of an open interval and two different terminals as {0, (0, 1), 1}. The difference between lower approximation and upper approximation sets is the indeterminate boundary whether the sufficient information of objects can classify the class belongs to. The more information of the equivalent class is classified, the more accurate objects description is.

Finally, the cloud model requires the partial membership with the random close intervals. The data distributes stochastically in the space as random [0, 1]. The randomness integrates with the fuzziness in order to conduct a model of mutual information between qualitative concepts and quantitative data.

Bionic Method

The Bionic method is the representative of Artificial Neural Network (ANN) and genetic algorithm. The neural network simulates a human brain as the networked patterns by a self-adaptive nonlinear dynamic system. The ANN contains many neurons which are connected with the immense and sophisticated junctions. The classification, clustering, and prediction GIS data of SDM with the complicate system and connected plentiful of neurons are successful by ANN. The ANN implements more precisely than the symbolic classification. The numerous neurons processing seems as a network that creates the non-linear function of a complex system. The information processing of neuron network involves with three layers which are input, middle, and output. The three layers operate depending on the dynamic response from the network status to the external data input. If between input and output layers have multiple hidden layers of units, ANN turns to be a deep neural network with the given method of deep learning. ANNs can reduce the noise disturbance in pattern recognition. So, the network is efficient on the high fault tolerance and robustness. If ANN is the nonlinear system

with many input variables, their convergence, stability, local minimum, and parameter adjustments of the network will be affected.

Others Related

- Rule Induction

The rule induction focuses on searching generic rule from the massive empirical data. The induction relates the basic statistical facts and instance of a big data while the deduction involves with axioms or basis reasoning on acknowledge knowledge (Clark and Niblet 1987). A concept tree and the rule induction which are the high-level patterns manipulate on uncovering the patterns among the spatial dataset.

- Decision Tree

The decision tree evolves the classification or decision set as a tree under certain spatial features. A test function on the tree structure of SDM generates the training object sets. Every created branch sets contain lower-nodes and sub-branches which generate iteratively. The leaf nodes are possible to be positive and negative examples.

- Visualization Techniques

In SDM, the spatial data and knowledge are better to explore and present visually. The form of visualization is worth than the words. The abstract patterns, relations, and tendencies of spatial data can be converted into the visualization by the computerized mechanism. The final results of visualization can be clarified by charts, maps, animations, graphics, images, trees, histograms, table, multimedia, and data cubes.

SDM Software

The SDM software implements the functions for discovering the spatial knowledge. The spatial data is truly existent the immense volume in the database. The implementation techniques are made use to instruct the computer understand

what is knowledge to find and how to find the spatial knowledge. The SDM software has to manipulate with the data mining system. The decision-making system is supported by the extraction of high level information. The Knowledge Discovery and data mining are invented as a technique to reveal the strategic information hidden.

Not all tools can support the knowledge discovery and data mining, the proper tool will be selected based on the data characteristics. As a feature classification scheme, was purposed and studied by Goebel and Gruenwald (1999). The researchers extracted the features of 43 different tools based on 3 groups that are general characteristics, database connectivity, and data mining characteristics. The results showed that first 2 groups describe the ability of each tool supports what kind of data. Then the third group showed that rule induction, decision trees, and statistical methods are the standard data mining techniques while the fuzzy and rough sets or genetic algorithms started developing on the knowledge discovery software.

The software can interact with the database and data warehouse. After that, the distributed and heterogeneous data are needed to operate from offline-based to online-based (cloud computing). The technology of Google or Baidu have to accelerate the services for end-users which support the location-based, positioning, recommendation and manufacturing. The application started with toolkit that is ready-to-use software. Their functions provide the variety of data mining algorithms such as IBM Intelligent Miner, SPSS Clementine, SAS Enterprise Miner.

The SDM software for the spatial data mainly focuses on the GIS data and RS images. Li et al. (2016) studied and implemented the GIS-DBMiner for GIS data and the RSImageMiner for imagery data. The GISDBMiner supports the common structured data which is stored in the database. Users start by sending the knowledge discovery command and the signals to spatial DBMS which enable to access the spatial database. Then, the algorithm performs and returns the discovered knowledge as the output. For the RSImageMiner is powerful of

the data management and data mining on image features such as shape, color, texture and pattern. The RSImageMiner manipulates the RS imagery data which working on the mining process to discover the feature knowledge. Then, continue with the knowledge-based image classification, retrieval and object recognition (Li et al 2015).

Examples of Application

The methods and techniques in SDM are powerful to increase the quality of the discovered knowledge. As the mathematical foundation of those algorithms, the classification, clustering, and prediction on the spatial data mining are interesting. Koperski (1999) reviewed a two-step decision classification by first spatial predicates extraction with the relief algorithm of ML. Both spatial (which refers to the data of geographic location, size and shape objects on the planet) and non-spatial (which refers to a creation of text, image, multi-media data for linking with the spatial data to specific the location) (Diwakar 2013) predicate the integrated with knowledge of classification decision. Then, the regional classification rules with a fuzzy decision tree in an object-oriented spatial database are examined (Marsala and Bigolin 1998). Moreover, the rough set applications also apply the various fields of approximate reasoning, ML, AI, pattern recognition, and knowledge discovery. The further improvement of rough set can be continued with the studying of the field of geospatial information sciences such as geo rough space.

The developed SDM applications focus on the behavioral data in the system due to the continuous growth in volume of the spatial data. The Internet-based geographic information (Goodchild 2007) and location-based sensing services are important to update the spatial information. The spatial data is employed in the electrical maps, planning networks, land use construction, and protection on farmland. Hence, the making use of GPS, GIS, remote sensing, radar, infrared computerized tomography imaging, etc. can be the merging techniques to rise up the application performance.

Future Directions for Research

From the SDM concepts and techniques mentioned above, there are previous researches that studied and applied the spatial data on geographical information to solve problems of resources changes (Berger 2001), environment changes (Mannion 1995), and land-use allocation effect (Carsjens and Van Der Knaap 2002). Hence, the possible future directions for research can adjust to an agriculture aspect. The trend of agricultural product is advertent and proposed. Since the main occupation in the agricultural countries works on the cultivation which can crop yearly. The mass volume of agricultural products is continuous growth. Then, the economic trend of each country and the world in both short and long-term can be driven by those farm produce. The product's popularity is the major impact for forecasting the demand and supply in the real market. If the quantity of agricultural product can be predictable, it definitely benefits to various clients. In term of investor, they can consider on the investment. While in term of government, they can determine the policy direction to support the agriculturists.

Cross-References

- [Big Data Analysis for Social Good](#)
- [Big Data Analysis Techniques](#)
- [Data Cleaning](#)
- [Spatial Data Integration](#)

References

- Berger T (2001) Agent-based spatial models applied to agriculture: a simulation tool for technology diffusion, resource use changes and policy analysis. *Agric Econ* 25(2–3):245–260
- Carsjens GJ, Van Der Knaap W (2002) Strategic land-use allocation: dealing with spatial relationships and fragmentation of agriculture. *Landsc Urban Plan* 58(2):171–179
- Clark P, Niblet TT (1987) The CN2 induction algorithm. *Mach Learn* J 3(4):261–283

- Diwakar S (2013) Spatial vs non spatial. <https://www.slideshare.net/SumanDiwakar/spatial-vs-non-spatial>. Publish on: 14 Apr 2013
- Ester M, Frommelt A, Kriegel HP, Sander J (2000) Spatial data mining: database primitives, algorithms and efficient DBMS support. *Int J Data Min Knowl Discov* 4(2):193–216
- Goebel M and Gruenwald L (1999) A survey of data mining and knowledge discovery software tools. *ACM SIGKDD explorations newsletter* 1(1):20–33
- Goodchild MF (2007) Citizens as voluntary sensors: spatial data infrastructure in the world of web 2.0. *IJSDIR* 2:24–32
- Han JW, Kamber M, Pei J (2012) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann Publishers Inc., Burlington
- Koperski K (1999) A progressive refinement approach to spatial data mining. PhD thesis, Simon Fraser University, British Columbia
- Li DR, Cheng T (1994) KDG-knowledge discovery from GIS. In: Proceeding of the Canadian conference on GIS, Ottawa, pp 1001–1012
- Li DY, Du Y (2007) Artificial intelligence with uncertainty. Chapman and Hall/CRC, London
- Li D, Wang S, Li D (2015) Spatial data mining: theory and application. Springer, Berlin/Heidelberg
- Li D, Wang S, Yuan H, Li D (2016) Software and applications of spatial data mining. *Wiley Interdiscip Rev Data Min Knowl Disc* 6(3):84–114
- Mannion AM (1995) Agriculture and environmental change: temporal and spatial dimensions. Wiley, Chichester
- Marsala C, Bigolin NM (1998) Spatial data mining with fuzzy decision trees. In: Ebecken NFF (ed) Data mining. WIT Press, Boston, pp 235–248
- Piatetsky-shapiro G (1994) An overview of knowledge discovery in databases: recent progress and challenges. In: Ziarko Wojciech P (ed) Rough sets, fuzzy sets and knowledge discovery. Springer, London, pp 1–10

Spatial Graph Big Data

Shashi Shekhar and Jayant Gupta
Department of Computer Science, University of Minnesota, Minneapolis, MN, USA

Synonyms

[Spatial networks big data](#); [Spatiotemporal graphs big data](#); [Spatiotemporal networks big data](#)

Definitions

Digital modeling of real-world networks (e.g., road networks, river network) to accurately represent geographic information is done using spatial graphs. Spatial graphs can represent n-ary relationships to model complex relations in the network. They differ from existing geographical models that can only represent binary relationships. Spatial graph is formally defined using the concepts of Xnodes and Xedges and Xgraphs in the following paragraph.

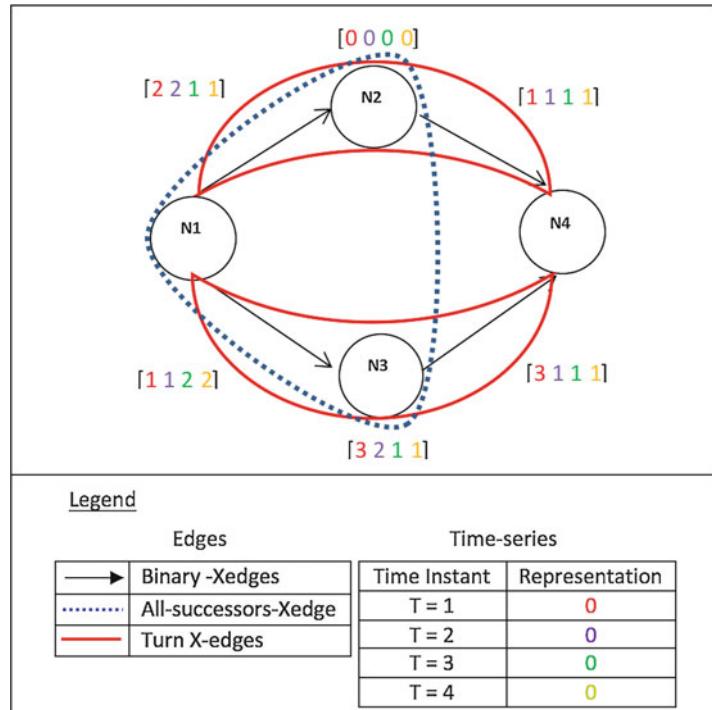
Xnode represents a real-world network feature (e.g., road intersection) that can have scalar or structured values (e.g., an array of scalars).

Xedge is a tree of Xnodes that can have scalar or structural values. Further, Xedge can be classified based on the type of network features being modeled (e.g., turn-Xedges) as shown in Fig. 1.

Xgraph can be defined as a set of Xnodes and a set of Xedges. **Spatial graph** is an ensemble of Xgraphs, which can represent n-ary relationships or discontinuity in a real-world network. Spatial graphs differ from hypergraphs because it consists of Xedge that can have a tree structure, whereas hypergraphs do not adhere to any specific structure.

Spatial graph big data is defined as a dataset with the following properties: (a) it describes the attributes of an Xgraph including attributes of its Xnodes, Xedges, and other substructures and (b) its volume, velocity, or variety exceeds the capacity of current data platforms. Examples include the longitudinal traffic volume associated with transportation networks such as road maps, as well as routes of airlines, buses, trains, ships, etc. as detailed in section “[Examples of Application](#)” on applications.

Figure 1 depicts a spatial graph big data with four nodes representing road intersections, namely, N1, N2, N3, and N4. The graph has seven Xedges including four binary Xedges ($N1 \rightarrow N2$), ($N2 \rightarrow N4$), ($N1 \rightarrow N3$), and ($N3 \rightarrow N4$) representing road segments connecting adjacent road intersections. The graph has an all-successor Xedge ($N1 \rightarrow (N2, N3)$) to represent the group of binary edges ($N1 \rightarrow N2$) and ($N1 \rightarrow N3$) from Xnode N1 to its children Xnodes N2 and

Spatial Graph Big Data,**Fig. 1** Example of spatial graph big data

N3. It also has two turn-Xedges ($N1 \rightarrow N2 \rightarrow N4$) representing a right turn and ($N1 \rightarrow N3 \rightarrow N4$) representing a left turn. The time series associated with the turn-Xedges represent turn delays. In this example, the right turn (i.e., $N1 \rightarrow N2 \rightarrow N4$) has no wait. However, the left turn has a wait of 3 for start-time 1, 2 for start-time 2, 1 for start-time 3, and no wait for start-time 4. Note that, in the figure, there are seven Xedges but six time series, because all-successor Xedges do not have a time series associated with it. For brevity “spatial graph” term is used in place of “spatial graph big data” in the following text.

Overview

City development aims to connect existing resources to ensure a productive environment for prospective citizens. Historically, the urban development has been driven by the stability of the major roadways that form the backbone of the urban infrastructure (Strano et al. 2012). In modern times, urban dwellers and goods are increasingly connected by multiple networks sup-

porting different means of transportation. The road networks have been augmented by rail networks (intercity, intracity), airport networks, and the port networks found located in cities near major rivers or navigable water bodies. These networks heavily influence many aspects of modern society, and they are implicated in many modern problems such as disease spread, congestion, and urban sprawl (Barthélemy 2011). Therefore, it is important to know how to model and analyze these types of transportation networks. In this regard, technology plays an important role to model the networks and understand their utilization.

Technological advancements, especially in the field of satellite systems, the Internet, and smartphones have had a major impact on the use and modeling of urban networks. GPS (Masumoto 1993) one of the Global Navigation Satellite System (GNSS) technologies (Lechner and Baumann 2000), is now used extensively for navigation on road networks. Further, Internet accessibility to smartphone users has been made possible by the advancement in electronics. Furthermore, when monitored at regular intervals, the location data from these devices allows us to digitally map

various movement activities (e.g., travel routes) to their physical locations, using map-matching techniques (Jensen and Tradišauskas 2009) for every user to perform different types of analysis (e.g., generating historical profiles of driver behavior).

Proliferation of technologies and the location data they generate is growing exponentially. In 2015, there were already around 2.6 billion smartphone subscriptions globally, and this number will continue to rise in the future especially in developing countries where the market is still maturing (Lunden 2015). In addition, commercial vehicles are using more sophisticated GPS devices that can monitor a variety of engine parameters (e.g., fuel usage, speed, etc.) in addition to location. Therefore, it is necessary to build technologies which can accurately and effectively model location-based data that are ever-growing size, variety, and update rate. In a big data paradigm, these issues correspond to the standard three Vs, i.e., volume, velocity, and variety (Zikopoulos and Eaton 2011). New database management systems are required to effectively harness these datasets.

In general, database applications are modeled using a three-step design process beginning with a conceptual data model (e.g., entity-relationship model), a logical data model (e.g., relational schema), and finally a physical design (e.g., R-Tree). To handle spatial features, the entity-relationship (ER) models were modified to pictogram-enhanced ER (PEER) to improve the geographical representation of the data in the model. The relational schema was modified by incorporating appropriate data types from SQL3 onward and translating the spatial relationship into spatial integrity constraints (Shekhar et al. 2004). Spatial graphs represent the physical modeling phase for modeling spatial network data, where, traditional graphs were modified to be-

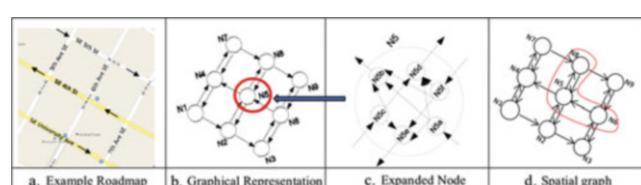
come a spatial graph to represent complex network properties with scalar or structural values.

To illustrate the limitations of simple graphs while modeling turns, consider the example road map as shown in Fig. 2a. Figure 2b shows a graphical representation of the road map where every node represents a road intersection and every directed edge represents a left or right lane of the road segment. In the original graph, the right turn at node N5 can be modeled as $N_6 \rightarrow N_5 \rightarrow N_8$, which requires the use of three nodes and two edges (3N, 2E). Consider, Fig. 2c, where the node N5 is expanded to six sub-nodes to model the turns at the intersection. Now the turn can be represented using $N_{5a} \rightarrow N_{5e}$, i.e., with just two sub-nodes and one edge, and no information is required from the other intersections. It is useful to be able to model turns at an intersection independent of other intersections. Finally, consider Fig. 2d that shows spatial graphs of segments and turns as turn-Xedges to represent the road map. The red-colored Xedge represents the turn; thus, with spatial graph, the turns can be modeled using a single Xedge.

Spatial graphs have been extensively studied in three main areas, namely, data modeling, database storage, and data mining algorithms. The research in spatial graph modeling is motivated to develop simple and effective representations of the data. For example, network size can grow rapidly as new nodes are added, and a simple model should easily accommodate and adapt to the changes. The spatial graph research in data storage is motivated to provide efficient graph-based operations (e.g., node retrieval, edge retrieval). The operations require data related to the same route; therefore, techniques (Evans et al. 2010) that store data on routes together helps to reduce the number of data page access. The research in the area of data mining uses spatial models to solve problems such as finding shortest

Spatial Graph Big Data,

Fig. 2 Modeling turns
(Shekhar et al. 2012) [Best in Color]



cost (e.g., shortest travel time) to different destinations or route planning to maximize traffic flow or crowd movement toward a destination (George et al. 2007). Section “Research Areas” describes each of these domains in further detail. Readers interested in example applications or future directions may consult section “Examples of Application” or section “Future Directions of Research” of this entry, respectively.

Research Areas

Modeling Spatial Graphs

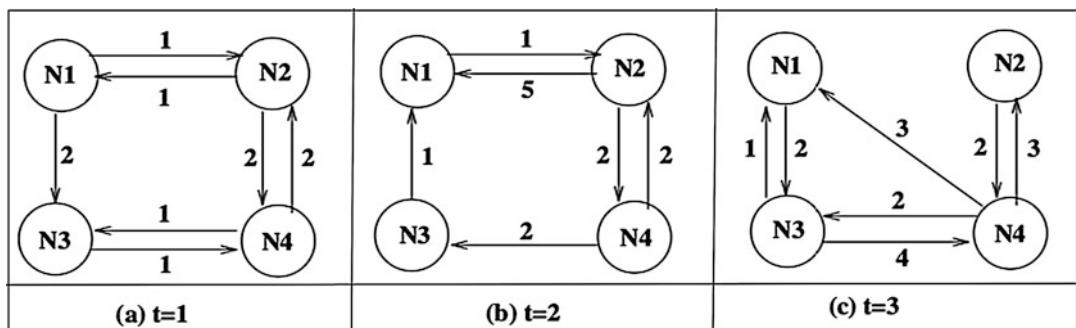
Spatial graphs typically represent transportation networks that can be viewed from one of two frames of reference, Eulerian or Lagrangian (Batchelor 2000). In the Eulerian frame of reference, traffic is observed as it travels past specific locations in the space over a period of time. It is similar to sitting on the side of a highway and watching the traffic pass a fixed location. In the Lagrangian frame of reference, the observer follows an individual moving object as it moves through space and time. This can be visualized as sitting in a car and driving down a highway. The following text discusses three spatial graph models: the snapshot and the time-aggregated graph (TAG) models, which use the Eulerian frame of reference, and the time-expanded graph (TEG) model, which uses the Lagrangian frame of reference (George et al. 2007).

1. Snapshot Model

A snapshot model represents a spatial graph with temporal attributes using a finite set of nodes and a finite set of edges connecting the nodes. Temporal attributes are represented by discrete time steps, each represented by a snapshot as shown in Fig. 9. The figure shows three time steps, $t = 1$ to $t = 3$, and shows the effect of temporal change on an edge, indicating travel time. For example, at time $t = 1$, edge N2-N1 has a value of 1. In the next time step, $t = 2$, the edge value increases to 5, indicating an increased travel cost for the edge.

2. Time-Expanded Graphs

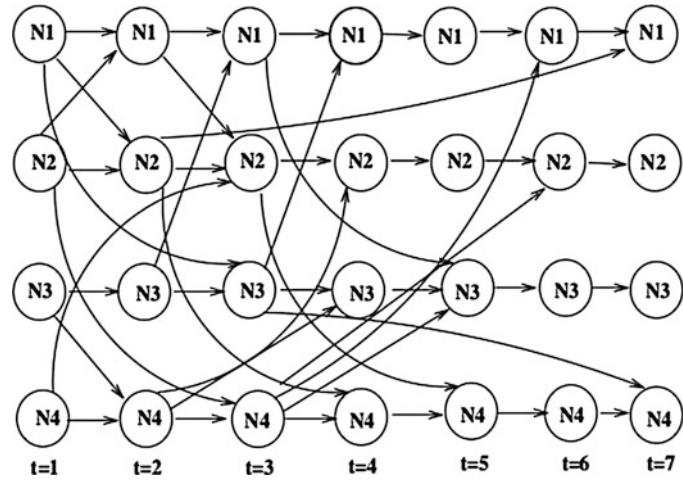
A time-expanded graph (TEG) (Köhler et al. 2002) model replicates each node along the time series such that a time-varying attribute (e.g., travel time) is represented between replicated nodes. Figure 4 shows the TEG corresponding to the spatial graph displayed earlier (Fig. 3). From the figure, observe that for edge N1-N2, weight of 1 is represented by an edge from N1 at $t = 1$ to N2 at $t = 2$. Further, the recurrence of similar edge across time between N1 and N2 at $t = 2$ mimics the invariance of edge weight over this time period. It is also interesting to see that N2 at $t = 1$ has an edge to N1 at $t = 2$ and that N2 at $t = 2$ has an edge to N1 at $t = 7$. This is because the edge weight for N2-N1 changes from 1 at $t = 1$ to 5 at $t = 2$ (Fig. 4).



Spatial Graph Big Data, Fig. 3 Snapshot model (George et al. 2007)

Spatial Graph Big Data,

Fig. 4 Time-expanded graph (George et al. 2007)

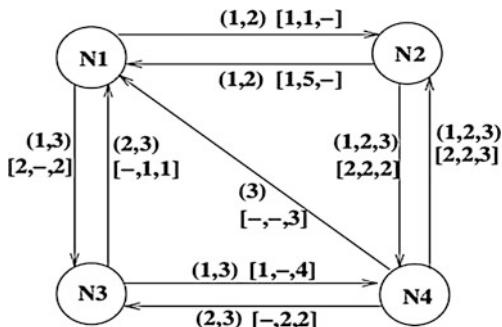


3. Time-Aggregated Graphs

A time-aggregated graph (TAG) stores a spatial graph with temporal attributes as the tuple $\{N, E, \text{Map}(N, TS_N), \text{Map}(E, TS_E), W\}$, where N is a set of nodes, E is a set of edges (E), $\text{Map}(N, TS_N)$ is the mappings from nodes to the time series associated with the nodes, $\text{Map}(E, TS_E)$ is the mappings from edges to the time series associated with the edges, and W is the time-dependent weights (e.g., travel time). Figure 5 shows the corresponding TAG for the networks as shown in Figs. 3 and 4. Observe that that node N1 has two time series associated with it for edge N1-N2 (1, 2) and N1-N3 (1, 3). Then, observe that the time-dependent weights for these edges are $[1, 1, -]$ and $[2, -, 2]$, respectively. The “-” symbol represents the time $t = 3$ when the edge N1-N2 does not exist.

Storage of Spatial Graphs

Large-scale spatial graphs with temporal attribute have hundreds of thousands of nodes and millions of time steps, amounting to terabytes of data. It is appropriate therefore to focus on secondary techniques for the storing of time-attributed spatial graphs for database systems. Figure 6 shows the process of storing a spatial graph. These methods make use of data files consisting of data pages and an indexing method. These methods employ an indexing method to output a data file containing the spatial graphs partitioned across



Spatial Graph Big Data, Fig. 5 Time-aggregated graph (George et al. 2007)

a set of data pages. Incorporating temporal data into a spatial graph poses significant challenges to their storage and analysis of disk I/O. Network topology and temporal access patterns add further constraints to the accessibility of data records.

One solution is to physically store topologically related nodes in the same data page, reducing the cost of data-page retrieval. Further, as the storage size decreases, the I/O cost is reduced due to a fewer number of pages containing the same data. The following texts describe three methods for storing spatial graphs: snapshot partitioning, longitudinal partitioning, and non-orthogonal partitioning. First, however, it is necessary to explain the concept of orthogonal storage vs. non-orthogonal storage, also known as synchronous time grouping vs. asynchronous time grouping.



Spatial Graph Big Data, Fig. 6 Process of storing spatiotemporal networks (STN)

Synchronous vs. Asynchronous Time Grouping

Synchronous time grouping is the clustering of data within a set time series, whereby some numbers of nodes and edges are stored within a set time interval. Each page is a snapshot, and longitudinal scheme represents a continuous time interval, either one time step or the entire time series. An example is storage of A1, B1, C1, and D1 in the same data page for snapshot partitioning as shown in Fig. 7a and storage of A1, A2, A3, and A4 in the same data page for longitudinal partitioning as shown in Fig. 7b. Asynchronous time data grouping allows storing data for disjoint time intervals. Thus, data from different time intervals are grouped and stored on the same data page. This model is known as sub-node model. It is useful for queries that retrieve the traversal times beginning at each time instant where temporal data is mostly not accessible orthogonally. An example is storage of A1, B2, C3, and D4 in the same data page as shown in Fig. 7c for Lagrangian-connectivity partitioning.

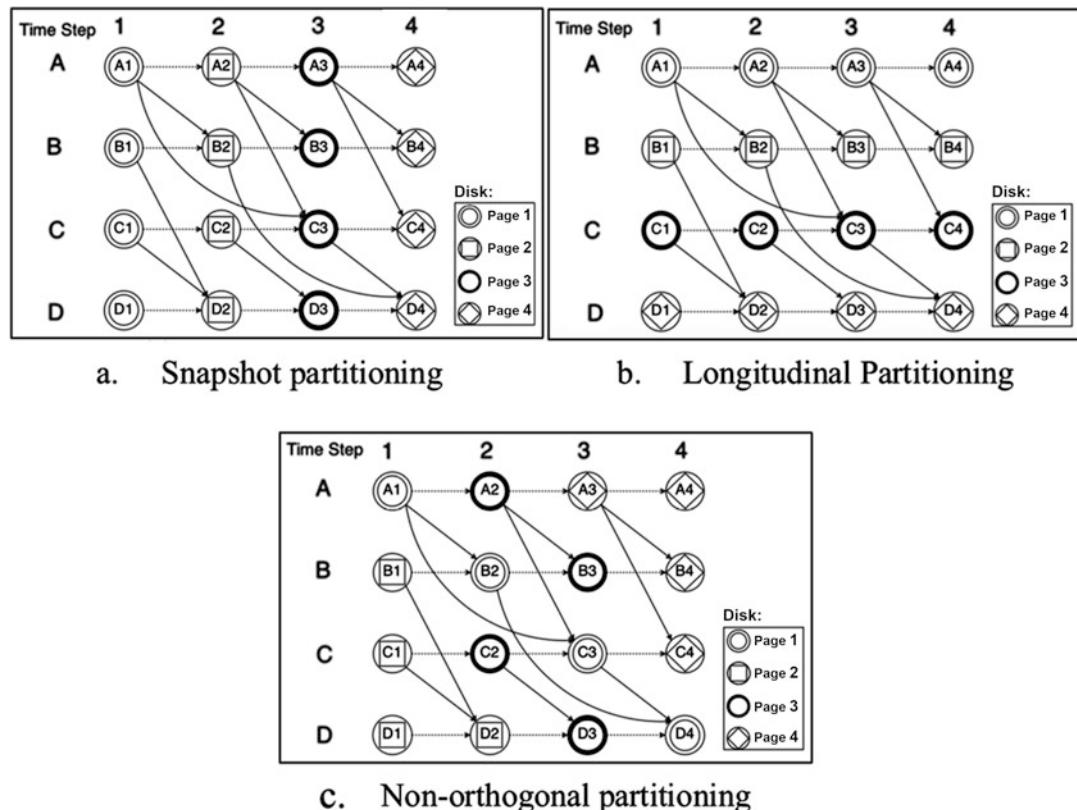
1. Snapshot Partitioning

Snapshot partitioning stores the data in pages using quad tree-based geometrical structures (e.g., R-Trees, R+-Trees, etc.). Figure 7a shows an example of snapshot partitioning, where a spatial graph with temporal attributes is represented as a snapshot model partitioned across the disk pages. As can be seen, the model preserves the graph state for a given time instant (say time $t = 1$). Further, the graph states are strictly partitioned based on time; thus, when traversing through the graph across time,

multiple-disk access needs to take place. For example, if a car's route has to be evaluated from A to D via C starting from time $t = 1$, it will require an access to edge AC at $t = 1$ stored in data page 1 and then access to edge CD at $t = 3$ stored in data page 3. Thus, the evaluation requires access to two data pages.

2. Longitudinal Partitioning

Longitudinal partitioning stores spatial graphs with temporal attributes based on the storage structure of the adjacency-list main memory. Figure 7b shows the longitudinal storage. As can be seen, each node is stored with its attribute information and all outgoing edges with their attribute information as an adjacency list. This orthogonal storage solution suffers from the increasing disk I/O to evaluate routes in large spatiotemporal networks. The example network has a short time series compared to its graph size, allowing multiple node records (with adjacency list) to fit inside a data page. However, if the time-series length was larger, then all the nodes may need multiple pages for storage. This is due to the long time series being stored with each node, resulting in only a small number of storable nodes on each data page. Consider the evaluation of the route ACD using longitudinal partitioning. First it requires accessing the traversal time attribute of edge AC at $t = 1$, stored on data page 1. Then, it requires accessing edge CD at $t = 3$, stored on data page 3, to complete the evaluation. Thus, for longitudinal partitioning, the evaluation requires two data pages.



Spatial Graph Big Data, Fig. 7 Spatial graph storage techniques (Evans et al. 2010)

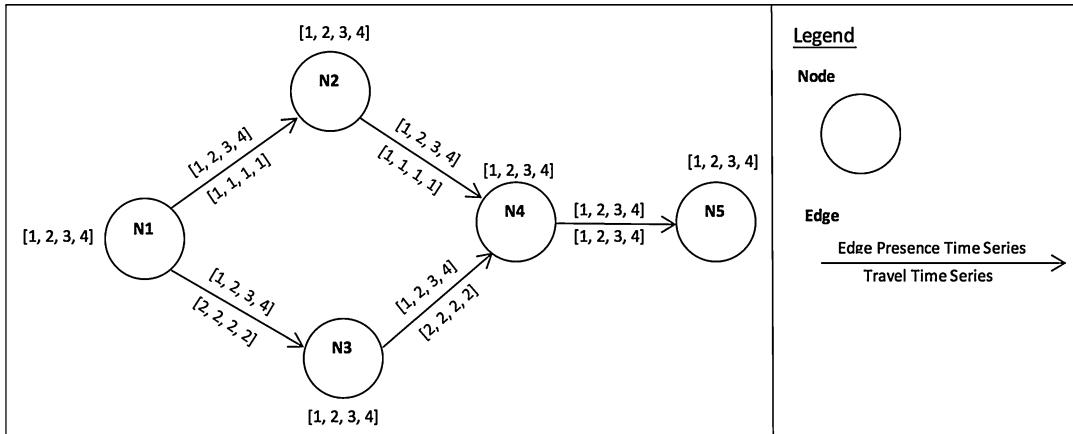
3. Lagrangian-Connectivity Partitioning

Lagrangian-connectivity partitioning (LCP) uses time-expanded graphs (TEG) to store spatial graphs with temporal attributes. Each partition stores the non-orthogonal patterns of route evaluation operations along with a novel data record based on sub-nodes. By representing a spatiotemporal network as a modified time-expanded graph, focusing on the Lagrangian connections between nodes (movement edges), a min-cut graph partitioning algorithm creates partitions clustering nodes by minimizing the cuts of these movement edges. This allows the algorithm to collocate connected temporal nodes together on data pages, stored as sub-node records as shown in Fig. 7c. This further reduces the I/O cost for the operations (described earlier). This can be seen by looking again at the evaluation of route ACD. Traversing from node

A1 to C3 and then C3 to D4 requires only one data page as all relevant sub-node records are on the same data page.

Data Mining Algorithms on Spatial Graphs

Spatial graphs with temporal attributes are used in emergency traffic planning and route-finding services. These applications require routes with the smallest duration for a given start time (useful for known events) and the route with the smallest duration that can occur at any of the start time. Developing efficient algorithms for finding such routes in a time-varying spatial network is challenging because these journeys do not always display a greedy property or optimal substructure, making techniques like dynamic programming inapplicable. The following text will outline algorithms for finding a route with the smallest duration for a given start time followed by an algorithm that finds routes with the smallest duration that can occur at any of the start times.



Spatial Graph Big Data, Fig. 8 Example spatial graph with temporal attributes

Spatial Graph Big Data, Table 1 Trace of the SP-TAG algorithm (George et al. 2007)

1. Shortest Path Computation for Time-Aggregated Graphs (SP-TAG)

The SP-TAG algorithm (George et al. 2007) uses the time-aggregated graphs to represent spatial networks, where each node (road intersection) has a node presence time series and each edge (road segment) has an edge presence time series annotated with travel time. The algorithm keeps track of the earliest time a node can be reached and updates the values as optimal values are found. The algorithm assigns t_{start} to the first node and puts the node in a priority queue (Cormen 2009). The steps are repeated until the priority queue is empty. At each step, the lowest-cost node is extracted from the queue. Then, the values for all its adjacent nodes are updated, and the extracted node is marked as closed. For each update, the minimum between the previous value and the new value of *current time + traversal time* is chosen. Then, the adjacent node is added to the queue. The time complexity of the SP-TAG algorithm is $O(m(\log T + \log n))$, where T is the number of nodes and m is the number of edges in the time-aggregated graph.

Figure 8 shows an example spatial graph having nodes with node presence time series and edges with edge presence time series, travel time series. Table 1 shows the SP-TAG algorithm trace on this graph, which shows the computation of

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	∞	∞	∞	∞
2	1	2 (closed)	3	∞	∞
3	1	2	3 (closed)	3	∞
4	1	2	3	3 (closed)	6
5	1	2	3	3	6 (closed)

the shortest path from N1 to N5. Each row represents an iteration of the algorithm that updates the values to newly found optimal values. For example, at iteration 2, node N2 is closed, and the N4 gets updated as shown in iteration 3.

2. Best Start Time Shortest Path (BEST) Algorithm

The BEST algorithm uses time-aggregated graphs to represent the network, where every node has a node presence time series and every edge has an edge presence time series and travel time series. In the output, each node has a time series, with the i th entry representing the current, least travel time to the destination node for the start time t_i . The algorithm uses an iterative label correcting approach (Ahuja et al. 1993), and each

entry in a node time series is modified according to the following condition:

$$C_u[t] = \min\{C_u[t], \sigma_{uv}(t) + C[t + \sigma_{uv}(t)]\},$$

where $uv \in E$,

$C_u[t]$ – Travel time from $u \in N$ to the destination for the start time t .

$\sigma_{uv}(t)$ – Travel time of the edge uv at time t .

The algorithm maintains a list of all nodes that change its cost according to the condition and terminates when there is no further improvement indicated by an empty list. The algorithm uses two-Q-based implementation (Pallottino 1984). The computational complexity of the BEST algorithm is $O(n^2mT)$, where n is the number of nodes, m is the number of edges, and T is the length of the time series.

Figure 9 shows an example with a figurative trace. At each step, the distance list from destination gets updated to show the shortest distance for all the time instants. Further, the parent pointer list gets updated to show the corresponding parent. Table 2 shows the trace for the algorithm showing shortest distance from N4 (destination) to all the nodes for each of the time instance. Further, the table shows the queue state. Note that in Fig. 9 edge presence time series and node presence time series are omitted because all the edges and nodes are present at all times.

Examples of Application

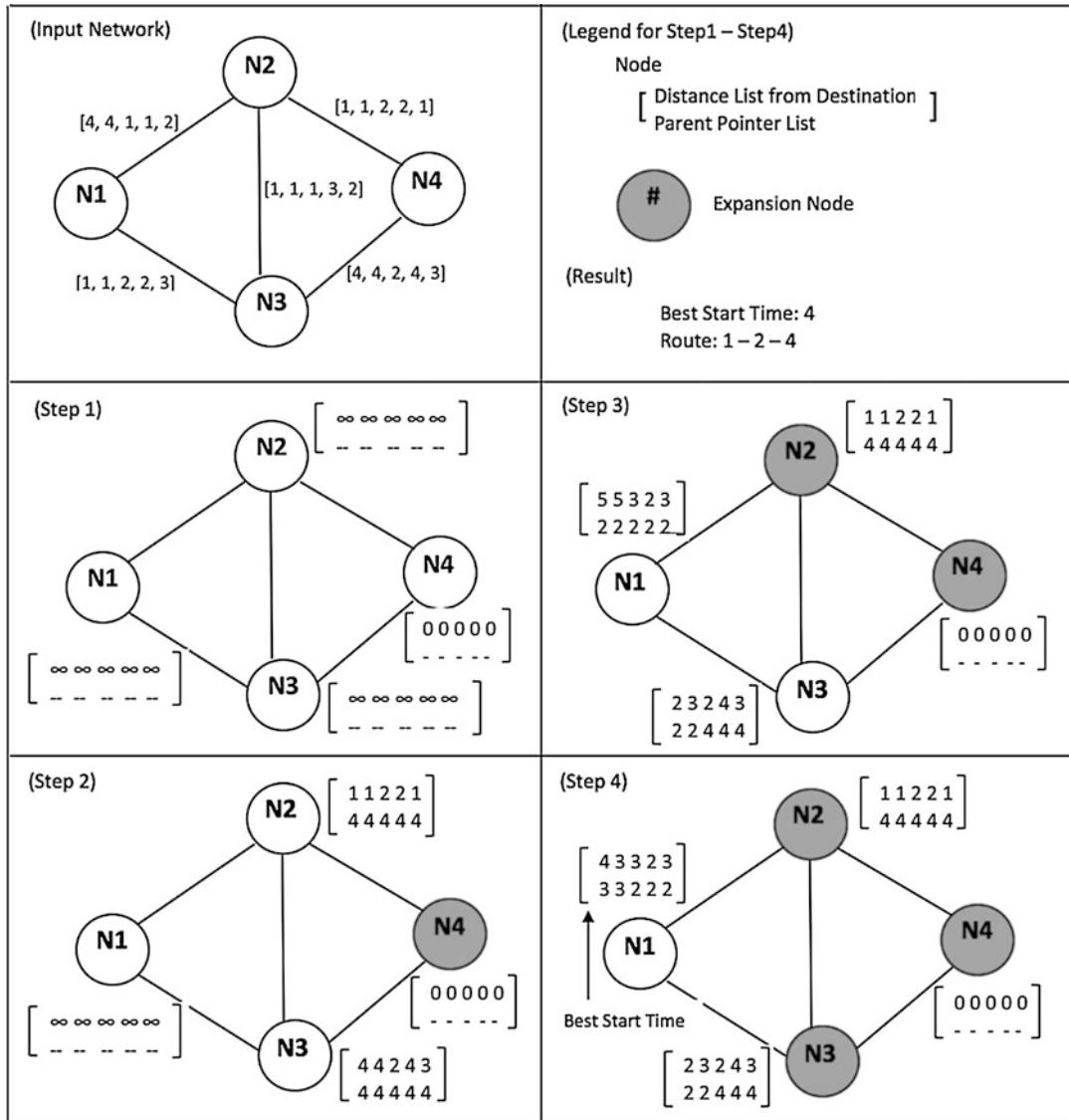
This section elaborates on three existing applications that rely on spatial graphs. Two of the applications show the use of different frames of reference (described in section “[Modeling Spatial Graphs](#)”) to the real-world applications. Section “[Historical Speed Profiles for Roadways](#)” describes the use of vehicle trajectory from a Eulerian frame of reference, and section “[GPS Trace Data for Improving Fuel Efficiency](#)” describes the use of vehicle trajectory from a Lagrangian frame of reference. Section “[Evacuation Route Planning](#)” discusses an important societal problem that talks about the use of spatial graphs in disaster management.

Historical Speed Profiles for Roadways

Traditionally, digital road maps have consisted of center lines and topologies of the road networks. These maps are used by navigation devices and web VTEQ, probe vehicles, and highway sensors to compile travel time information across road segments for all times of the day and week at fine temporal resolution (seconds or minutes). The profiles have data for every 5 min, which can then be applied to the road segment, building up an accurate picture of speeds based on historical data. An example distribution is shown in Fig. 10 (Shekhar et al. 2012). Such temporally detailed (TD) road maps contain much more speed information than traditional road maps. While traditional road maps have only one scalar value of speed for a given road segment (e.g., EID 1), TD road maps may potentially list speed/travel time for a road segment (e.g., EID 1) for thousands of time points (Fig. 10a) in a typical week. This allows a commuter to compare alternate start times in addition to alternate routes. It may even allow comparison of (different) start time and route combinations to select distinct preferred routes and distinct start times. For example, route ranking may differ across rush hour and non-rush hour and in general across different start times. However, TD road maps are big, and their size may exceed 10^{13} items per year for the 100 million road segments in the USA when associated with per-minute values for speed or travel time. Thus, industry is using speed profiles that are a lossy compression based on the idea of a typical day of a week, as illustrated in Fig. 10b, where each road segment and day of the week pair is associated with a time series of speed values for each hour of the day.

GPS Trace Data for Improving Fuel Efficiency

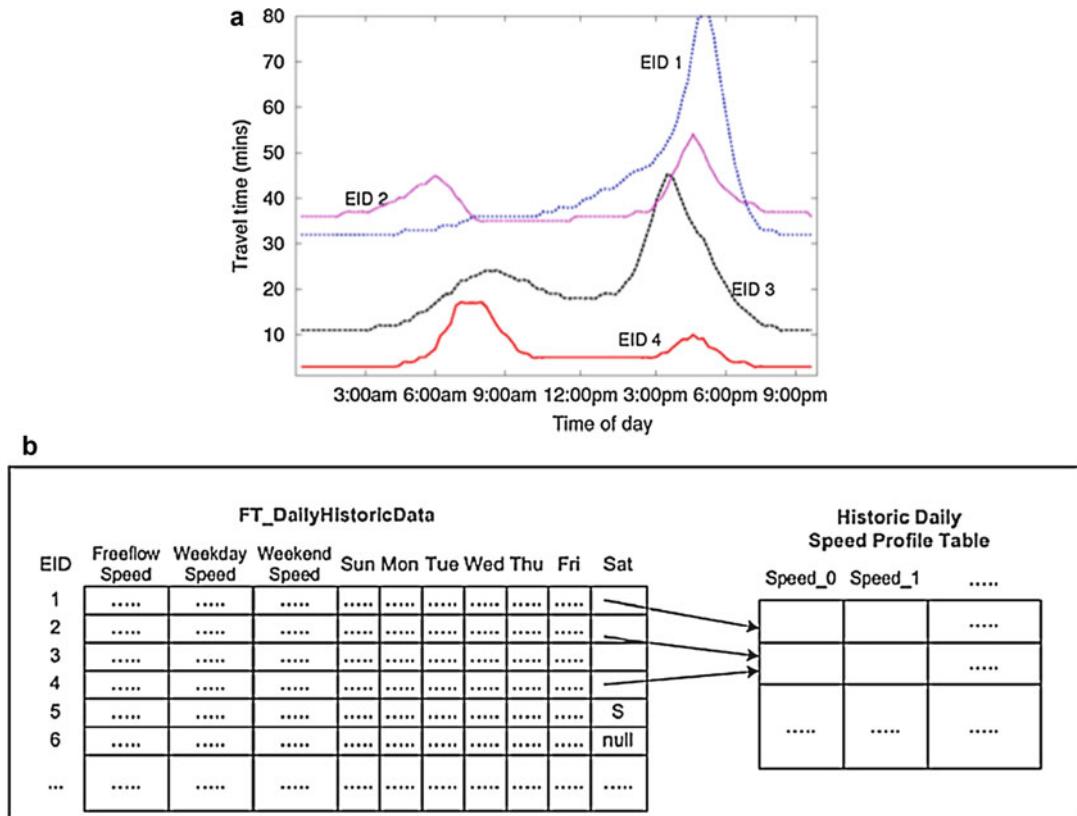
GPS trajectories are available for a large collection of vehicles due to rapid proliferation of cell phones, in-vehicle navigation devices, and other GPS data-logging devices such as those distributed by insurance companies. According to Shekhar et al. (2012), GPS traces allow indirect estimation of fuel efficiency and GHG emissions via estimation of vehicle speed, idling, and con-



Spatial Graph Big Data, Fig. 9 Trace of the BEST algorithm (George et al. 2007)

Spatial Graph Big Data, Table 2 Trace of the BEST algorithm (George et al. 2007)

Iteration	N1	N2	N3	N4	Queue
1	$\infty \dots \infty$	$\infty \dots \infty$	$\infty \dots \infty$	[0, 0, 0, 0, 0]	N1
2	$\infty \dots \infty$	[1, 1, 2, 2, 1]	[4, 4, 2, 4, 3]	[0, 0, 0, 0, 0]	N2, N3
3	$\infty \dots \infty$	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N3
4	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N1
5	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	-

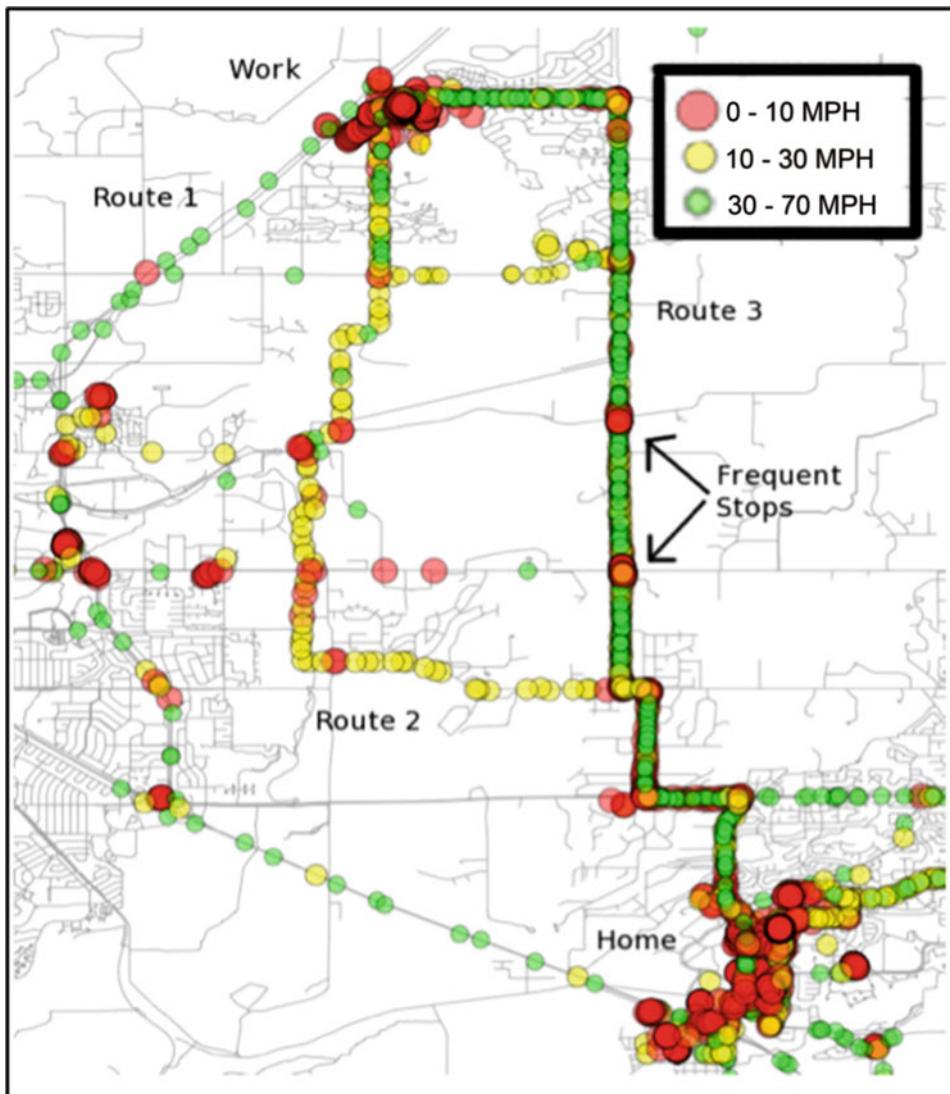


Spatial Graph Big Data, Fig. 10 Spatial big data on historical speed profiles (Shekhar et al. 2012)

gestion. They also make it possible to provide personalized route suggestions to users to reduce fuel consumption and GHG emissions. For example, Fig. 11 shows 3 months of GPS trace data from a commuter with each point representing a GPS record taken at 1 min intervals, 24 h a day, 7 days a week. As can be seen, three alternative commute routes are identified between home and work from this dataset. These routes can be compared for engine idling which are represented by darker (red) circles. Assuming the availability of a model to estimate fuel consumption from speed profiles, one may even rank alternative routes for fuel efficiency. Again, a key hurdle is the dataset size, which can reach 10^{13} items per year given constant minute-resolution measurements for all 100 million US vehicles.

Evacuation Route Planning

Large public gatherings require effective management to preserve public safety. One of the key responsibilities of a civil administration is the capability to manage adverse conditions (e.g., terrorist acts, accidents) that may require finding efficient routes to evacuate all or a portion of a population. The evacuation route planning problem (Yang et al. 2012) finds routes that minimize evacuation time given a transportation network, a population, and a set of observations. To be effective and timely, evacuation planning methods need to adhere to network capacity constraints and provide reasonable computation time. Evacuation route planning involves (taking) a big data perspective due to the size of transportation networks (order



Spatial Graph Big Data, Fig. 11 A commuter's GPS tracks over 3 months reveal preferred routes (Shekhar et al. 2012)

of 10^3 nodes and 10^6 edges), the large number of evacuees (order of 10^6), as well as the capacity constraints for every node and edge.

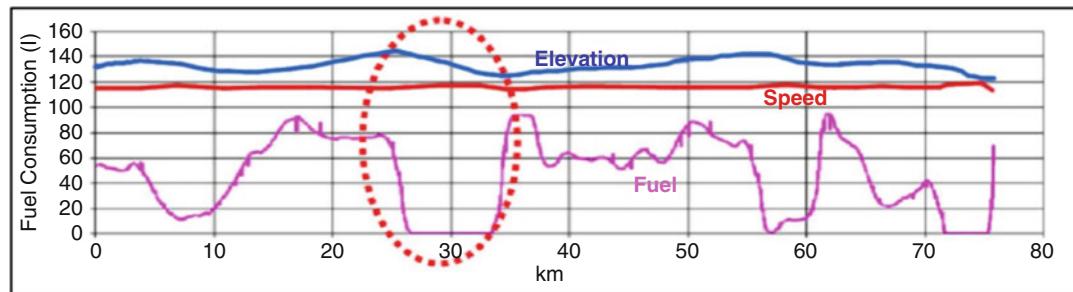
Future Directions of Research

The section describes two future directions of research related to spatial graphs. The first is driven by the increased use of sensors that can measure every aspect of vehicles including its

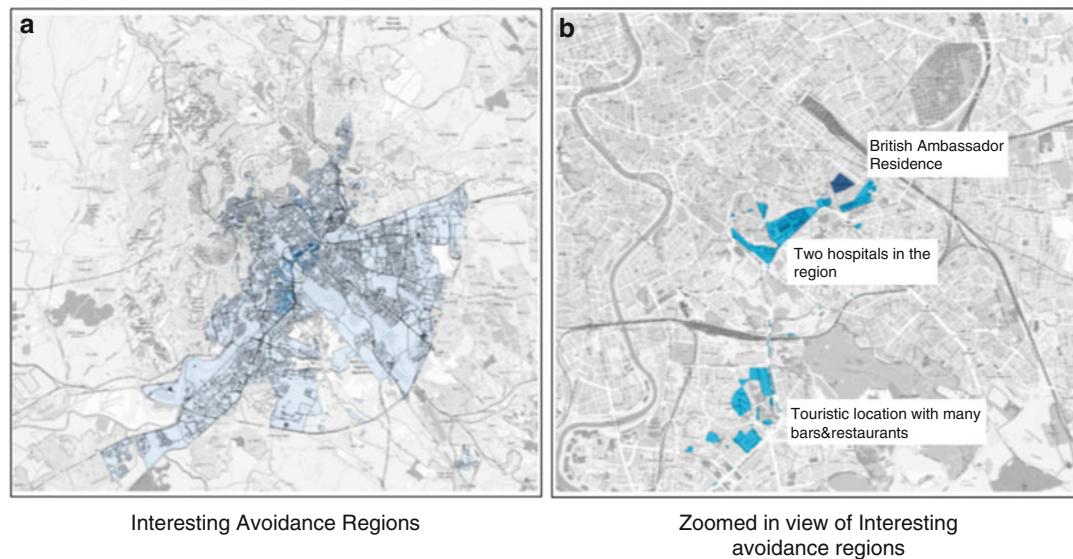
surroundings. This would potentially allow to make variety of profiles for a vehicle. The second is driven by the need to identify city areas that are affected due to multiple reasons (e.g., natural calamities, adverse neighborhood) based on trajectory data.

Time Annotated Engine Measurement Spatial Data

Engine measurement datasets may be used to study the impacts of the environment (e.g., ele-



Spatial Graph Big Data, Fig. 12 Engine measurement data improve understanding of fuel consumption (Capps et al. 2008)



Spatial Graph Big Data, Fig. 13 Case study on identification (Eftelioglu et al. 2018)

vation changes, weather), vehicles (e.g., weight, engine size, energy source), traffic management systems (e.g., traffic light timing policies), and driver behaviors (e.g., gentle acceleration or braking) on fuel savings and greenhouse gas emissions. Fuel efficiency can be estimated from fuel levels and distance traveled as well as engine idling from engine RPM. These attributes may be compared with geographic contexts such as (e.g., elevation changes) to improve understanding of fuel efficiency and greenhouse gas emission.

For example, Fig. 12 shows heavy truck fuel consumption as a function of elevation from a study (Capps et al. 2008) at Oak Ridge National Laboratory. Notice how drastically fuel consumption changes drastically with elevation

slope changes. Commercial fleet owners have studied such datasets to fine-tune routes to reduce unnecessary idling. It is tantalizing to explore the potential of these datasets to help consumers gain similar fuel savings and reductions in greenhouse gas emission. However, these datasets can grow big. Measurements of 10 engine variables, once a minute, over the 100 million US vehicles in existence, may have 10^{14} data items per year.

Distressed Area Identification

Distressed area identification uses GPS trajectories on a road network to identify distressed areas or the regions that are usually avoided by the commuters. This emerging area is important to applications such as sociology, city/transporta-

tion planning, and crime mitigation, where it can help domain users to understand the driver behavior under varying adverse conditions (e.g., rush hour, congestion, dangerous neighborhoods, etc.).

For example, Fig. 13 shows results from a recent case study (Eftelioglu et al. 2018) on a real dataset consisting of 1312 vehicle GPS trajectories in Italy over a period of 3 years. The study used the input trajectory data and road network data to output interesting avoidance regions as shown in Fig. 13a. The zoomed-in view (Fig. 13b) shows that drivers avoid areas with increased security measures (e.g., important government buildings) or increased congestion (e.g., hospitals or tourist locations).

Cross-References

► Spatial Data Mining

References

- Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Pearson. ISBN-13: 978-0136175490
- Barthélemy M (2011) Spatial networks. Phys Rep 499(1):1–101. Elsevier
- Batchelor GK (2000) An introduction to fluid dynamics. Cambridge University Press. ISBN 978–0521663960
- Capps G, Franzese O, Knee B, Lascurain MB, Otaduy P (2008) Class-8 heavy truck duty cycle project final report. ORNL/TM-2008/122
- Cormen TH (2009) Introduction to algorithms. MIT press. ISBN 978–8120340077. eMarketer, <https://goo.gl/TcMzQX>
- Eftelioglu E, Tang X, Shekhar S (2018) Avoidance region discovery: a summary of results. SIAM international conference on data mining (Accepted)
- Evans MR, Yang K, Kang JM, Shekhar S (2010) A lagrangian approach for storage of spatio-temporal network datasets: a summary of results. In: Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems. ACM, New York, pp 212–221
- George B, Kim S, Shekhar S (2007) Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results. In: Papadias D, Zhang D, Kollios G (eds) Advances in Spatial and Temporal Databases. SSTD 2007. Lecture Notes in Computer Science, vol 4605. Springer, Berlin/Heidelberg
- Jensen C, Tradišauskas N (2009) Map matching. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer, Boston
- Köhler E, Langkau K, Skutella M (2002) Time-expanded graphs for flow-dependent transit times. In: Möhring R, Raman R (eds) Algorithms – ESA 2002. Lecture notes in computer science, vol 2461. Springer, Berlin/Heidelberg
- Lechner W, Baumann, S. (2000). Global navigation satellite systems. Comput Electron Agric, 25(1), 67–85. Elsevier
- Lunden I (2015) 6.1B Smartphone Users Globally By 2020. Overtaking basic fixed phone subscriptions. <https://goo.gl/lcMcPK>
- Masumoto Y Global Positioning System (1993) U.S. Patent No. 5,210,540. U.S. Patent and Trademark Office, Washington, DC
- Pallottino S (1984) Shortest-path methods: complexity, interrelations and new propositions. Networks 14(2):257–267. John Wiley & Sons, Inc
- Shekhar S, Vatsavai RR, Ma X, Yoo JS (2004) Navigation systems: A spatial database perspective. as Chapter 3 in Location-Based Services, Agnes Voisard and Jochen Schiller. Elsevier, pp 41–80. ISBN 9781558609297
- Shekhar S, Gunturi V, Evans MR, Yang K (2012) Spatial big-data challenges intersecting mobility and cloud computing. In: Proceedings of the eleventh ACM international workshop on data engineering for wireless and mobile access. ACM, New York, pp 1–6
- Strano E, Nicosia V, Latora V, Porta S, Barthélémy M (2012) Elementary processes governing the evolution of road networks. Sci Rep 2. Nature.com
- Yang K, Gunturi VMV, Shekhar S (2012) A Dartboard Network Cut Based Approach to Evacuation Route Planning: A Summary of Results. In: Xiao N, Kwan MP, Goodchild MF, Shekhar S (eds) Geographic Information Science. GIScience 2012. Lecture Notes in Computer Science, vol 7478. Springer, Berlin/Heidelberg
- Zikopoulos P, Eaton C (2011) Understanding big data: analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media

Spatial Joins

► Query Processing: Joins

Spatial Joins in Clusters

► Query Processing: Joins

Spatial Networks Big Data

► Spatial Graph Big Data

Spatio-social Data

Mohamed Sarwat and Yuhan Sun

School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Synonyms

GeoSocial data; Spatial data

Definitions

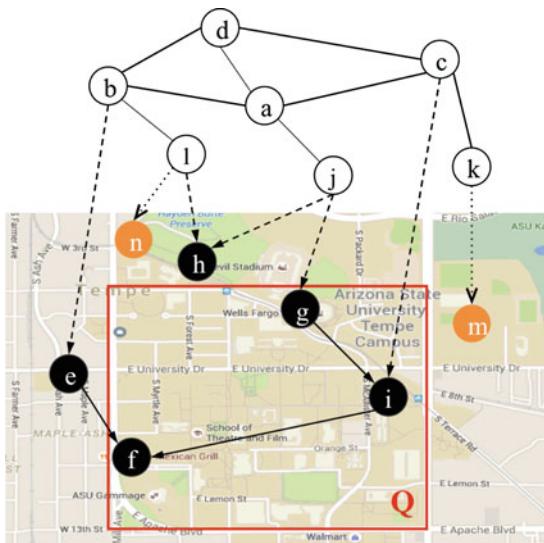
In the past decade, social networking services, e.g., Facebook and Twitter, managed to unprecedentedly connect hundreds of millions of people all over the world. Users register to online social networks in order to keep in touch with their friends and family, learn about their news, get recommendations from them, and engage in online social events. Thanks to the widespread use of mobile and wearable devices, popular social networks, e.g., Facebook, prompt users to add spatial attributes to social entities, e.g., check-ins, posts, and geo-tagged photos. Such spatial attributes are already being tied to social networking data to form what we call, *Spatio-Social Data*. Spatio-Social data brings both the physical space, social relationships, user interactions, and social media content into effect together, which led to the rise of many interesting applications. For instance, users in a location-based social networking service (e.g., Foursquare and Facebook Places) are associated with a geo-location and might alert friends when visiting a place (e.g., restaurant, bar) by checking in on their mobile phones.

Overview

Figure 1 gives an example of how Spatio-Social data looks like. The figure depicts a social graph in which there exist three types of entities: (i) *person* (e.g., Alice, Bob, Carol, Dan, etc.) with a name and age attributes, (ii) *place* (e.g., restaurants) with a name and a spatial location attributes, and (iii) *content* (e.g., tweet) with id, social media content (e.g., Tweet text), and location attributes. The graph consists of four types of social connections: (i) *friend of* that represents the social relationship between persons, e.g., Alice is a friend of Dan; (ii) *visited* that represents the users' visits to places, e.g., Dan visited Pita Jungle restaurant in Tempe AZ; (iii) *posted* that represents the relationship between people and social media content, e.g., Mat posted a tweet within the ASU stadium area; (iv) *rated* that represents the users' interactions with the physical space, e.g., Kate gave a five-star rating to Sushi 101.

Spatio-Social Graph

In a GeoSocial graph, a user may issue the following query: “Find Restaurants in Phoenix that are visited by Alice’s Friends.” In addition, such data can be utilized to process GeoSocial analytic tasks, e.g., “Report restaurants in Phoenix that are visited by teenagers ($13 < \text{age} < 18$).” We call such queries that search the social graph with both social and spatial predicates, *GeoSocial Graph Search (G^2S) queries*. We can formally define a GeoSocial Graph Search G^2S query as a regular path query with geospatial predicates performed on the social graph. A G^2S query takes as input a sequence of social graph predicates as well as a spatial predicate. The social predicate is applied to traditional social graph entities and/or connections with nonspatial attributes, e.g., person, whereas the spatial predicate is performed on social entities that possess spatial attributes, e.g., restaurants. G^2S then returns a set of social graph paths that match the social predicates as well as satisfy the spatial predicate. Existing graph database systems, e.g., Neo4j, allow users to define spatial properties on graph elements. MongoDB is utilized by Armenatzoglou et al. (2013)



- a: {name: Alice, age: 19}
 b: {name: Dan, age: 20}
 c: {name: Carol, age: 35}
 d: {name: Bob, age: 25}
 j: {place: Kate, age: 18, ...}
 k: {place: Mat, age: 23, ...}
 l: {place: Katharine, age: 21, ...}
- e: {place: Pita Jungle, location:<x1,y1>}
 f: {place: Chipotle, location:<x2,y2>}
 g: {place: Sushi 101,...}
 h: {place: Subway,...}
 i: {place: McDonald's,...}
- m: {id: 1, tweet: "Nice Weather", location:<x10,y10>}
 n: {id: 2, tweet: "Good Game", location:<x10,y10>}

Spatio-social Data, Fig. 1 GeoSocial graph data

to manage both social and geographic data. Each user in the GeoSocial graph is assigned a spatial location. A Range Friends query searches for friends within a given distance from the user. The Nearest Friend query searches the nearest friend of a user.

Geo-tagged Social Media

Social media content is rich in content, e.g., text, video, hyperlinks, photos, and spatial location. Existing work studied keyword search on streaming social media data (Busch et al. 2012; Wu et al. 2013; Yao et al. 2012). Other newly emerging applications on GeoSocial media include event detection (Abdelhaq et al. 2013; Li et al. 2012; Marcus et al. 2011; Watanabe et al. 2011), news extraction (Sankaranarayanan et al. 2009), and analysis (Tweet Tracker 2013). Bao et al. (2012) propose a GeoSocial news feed, which ranks the news feed based on both the social and geospatial proximity of each post to the user.

User Check-In Data

Myriad Web applications produce *location-based ratings* that embed user and/or item locations. For example, applications like Foursquare and Yelp allow users to “check in” at spatial destinations

(e.g., restaurants) and rate their visit and thus are capable of associating both user and item locations with ratings. Existing recommendation techniques assume ratings are represented by the (*user*, *rating*, *item*) triple and thus are ill-equipped to produce location-aware recommendations.

Recent systems produce *three* main types of location-based user interactions (Sarwat et al. 2014; Levandoski et al. 2012a): (1) Spatial ratings for nonspatial items, represented as a four-tuple (*user*, *ulocation*, *rating*, *item*), where *ulocation* represents a user location. Items are nonspatial in nature (e.g., movies) and thus do not have an associated location. As an example, traditional e-commerce applications (e.g., Netflix) may use a user’s home address as *ulocation*, while mobile applications may associate the location where the user rated the item as the *ulocation*. (2) Nonspatial ratings for spatial items, represented as a four-tuple (*user*, *rating*, *item*, *ilocation*), where *ilocation* represents an item location. Examples of applications that produce such ratings are e-commerce applications that gather user opinions on venues/destinations (e.g., restaurant review websites), but do not collect user locations.

Key Research Findings

Spatio-Social Graph Queries

Basic approaches that answer GeoSocial Graph Search (G^2S) queries fall into two main categories, listed as follows:

- (1) *Social-Then-Spatial (SoSpa)* approach This approach leverages the query processor of an existing graph data management system (Kylola et al. 2012; Prabhakaran et al. 2012; Sarwat et al. 2012, 2013b; Shao et al. 2013). It first traverses the social graph to find matching paths (Chen and Chen 2008; Fan et al. 2011; Jin et al. 2010). When a spatial entity is encountered during the social graph traversal, SoSpa tests these entities against the spatial predicate. Finally, SoSpa returns only paths that satisfy both the social path predicates and the spatial range predicate.
- (2) *Spatial-Then-Social (SpaSo)* approach This approach runs in two sequential phases:
 - (a) Spatial filtering: This phase employs a state-of-the-art spatial database management system that harnesses a spatial index, e.g., R-tree, to first retrieve the social graph entities that satisfy the spatial predicate.
 - (b) Social filtering: This phase traverses the social graph to only return social graph paths that match the social predicates and contains only the spatial entities retrieved in the spatial filtering phase. Even though both approaches correctly answer G^2S queries, nonetheless they may lead to performance penalties by traversing unnecessary graph paths that incur extra I/O and CPU overhead which increases the overall latency of G^2S queries (Sarwat and Sun 2017).

Spatio-Social Media Queries

Existing systems provide scalable indexing mechanisms that allow users to interactively explore spatio-textual data at scale. Various spatial and spatio-textual indexing techniques are explored in the literature (Budak et al. 2014; Magdy et al. 2014; Skovsgaard et al. 2014) to

support spatial queries on geo-tagged social media. Novel systems extend those techniques to reduce the indexing latency and increase the digestion rate. Existing research work addresses the problem of tuning the in-memory part of the spatio-textual index to minimize the overall index storage and maintenance overhead. Initial studies show that space-partitioning spatial indexes achieve better performance in processing real-time social media, e.g., spatial grid index or spatial quad tree. On the contrary to data-partitioning spatial indexes, e.g., R-tree, cell boundaries in space-partitioning indexes do not change with the incoming data. Instead, each cell covers a specific area of the space and holds whatever data it contains. Recent research work also studies the problem of real-time geo-tagged social media content insertion in the spatio-textual, spatio-visual index by employing lazy batch updates so that the amortized insertion cost per social media content is minimized. A large body of work investigates compression techniques to minimize the memory footprint of geo-tagged social media to efficiently utilize main memory and storage resources. Such systems investigate storing only necessary information in main memory, e.g., recent tweets or local top frequent keywords, and get rid of any other information so that incoming spatial exploration queries can fetch its answers in main memory and avoid hitting the relatively slower secondary storage.

S

Spatio-Social Recommendation

Existing spatial database systems allow users to look up spatial data that matches exactly the designated query and hence do not leverage the user interaction with the physical space. Recommendation (Sarwat et al. 2017; Levandoski et al. 2012b; Herlocker et al. 2004; Koutrika et al. 2009) is the process of suggesting useful data to the user based on a large pool of historical user interaction data (e.g., a user rating a venue on Yelp). To take into account the user interaction with the physical space, existing approaches extend spatial data structures with user interaction data to support spatial data recommendation. The straightforward approach constructs a spatial

index, e.g., R-tree, on all spatial data objects and calculates the similarity between different items and data points. When the user asks for a set of spatial objects in a certain boundary region, the system first searches for all the points lying within the region. It is not feasible to go through all the data point outcomes from the search. For this purpose, the system filters huge data ignoring all those points that may not interest him based on his previous interactions with the system. Then, only the items with top-k recommendation scores are displayed to the user. The performance can be further optimized to give faster recommendations by augmenting existing spatial indexes to filter spatial data using both their spatial attributes and their recommendation scores.

The system produces recommendations by employing an adaptive hierarchical grid structure to partition data ratings by their *user location* attribute into spatial regions of varying sizes at different hierarchies. For a querying user located in a region R , we apply an existing collaborative filtering technique that utilizes only the ratings located in R . Existing systems can dynamically balance scalability and recommendation locality. In this case, the system produces recommendations by using *travel penalty*, a technique that penalizes recommendation candidates the further they are in travel distance to a querying user (Sarwat et al. 2013a). The challenge here is to avoid computing the travel distance (Hjaltason and Samet 1999) for all spatial items to produce the list of k recommendations, as this will greatly consume system resources. Existing systems address this challenge by employing an efficient query processing framework capable of terminating early.

Future Direction for Research

Spatio-Social data is rich in semantics. The graph model can carry structure information, spatial information, textual information, and user data at the same time. Such data is inherently heterogeneous; hence there is always room for novel applications that combine a variety of Spatio-Social

aspects. Another direction of future research is to explore the appropriate approach to analyze and explore Spatio-Social data using visual map interfaces. The challenge is that Spatio-Social data is growing at a staggering rate. In that case, an interesting area of research will be to extend de facto distributed computing systems and streaming engines to support fast and scalable processing of Spatio-Social data.

Cross-References

- ▶ [Linked Geospatial Data](#)
- ▶ [Spatial Graph Big Data](#)

References

- Abdelhaq H, Sengstock C, Gertz M (2013) EvenTweet: online localized event detection from Twitter. In: VLDB
- Armenatzoglou N, Papadopoulos S, Papadias D (2013) A general framework for geo-social query processing. Proc VLDB Endow 6(10):913–924
- Bao J, Mokbel MF, Chow C-Y (2012) Geofeed: a location aware news feed system. In: 2012 IEEE 28th international conference on data engineering. IEEE, pp 54–65
- Budak C, Georgiou T, Agrawal D, Abbadi AE (2014) GeoScope: online detection of geo-correlated information trends in social networks. In: VLDB
- Busch M, Gade K, Larson B, Lok P, Luckenbill S, Lin J (2012) Earlybird: real-time search at Twitter. In: ICDE
- Chen Y, Chen Y (2008) An efficient algorithm for answering graph reachability queries. In: Proceedings of the IEEE international conference on data engineering, ICDE
- Fan W, Li J, Ma S, Tang N, Wu Y (2011) Adding regular expressions to graph reachability and pattern queries. In: Proceedings of the IEEE international conference on data engineering, ICDE
- Herlocker JL, Konstan JA, Terveen LG, Riedl JT (2004) Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst TOIS 22(1):5–53
- Hjaltason GR, Samet H (1999) Distance browsing in spatial databases. ACM TODS 24(2):265–318
- Jin R, Hong H, Wang H, Ruan N, Xiang Y (2010) Computing label-constraint reachability in graph databases. In: Proceedings of the ACM international conference on management of data, SIGMOD
- Koutrika G, Bercovitz B, Garcia-Molina H (2009) FlexRecs: expressing and combining flexible recommendations. In: Proceedings of the ACM international conference on management of data, SIGMOD, pp 745–758

- Kyrola A, Blelloch G, Guestrin C (2012) PowerGraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the USENIX symposium on operating system design and implementation, USENIX OSDI
- Levandoski JJ, Sarwat M, Eldawy A, Mokbel MF (2012a) LARS: a location-aware recommender system. In: Proceedings of the international conference on data engineering, ICDE
- Levandoski JJ, Sarwat M, Mokbel MF, Ekstrand MD (2012b) RecStore: an extensible and adaptive framework for online recommender queries inside the database engine. In: Proceedings of the international conference on extending database technology, EDBT
- Li R, Lei KH, Khadiwala R, Chang KC-C (2012) TEDAS: a Twitter-based event detection and analysis system. In: ICDE
- Magdy A, Mokbel MF, Elnikety S, Nath S, He Y (2014) Mercury: a memory-constrained spatio-temporal real-time search on microblogs. In: ICDE, pp 172–183
- Marcus A, Bernstein MS, Badar O, Karger DR, Madden S, Miller RC (2011) Twitinfo: aggregating and visualizing microblogs for event exploration. In: CHI
- Prabhakaran V, Wu M, Weng X, McSherry F, Zhou L, Haridasan M (2012) Managing large graphs on multi-cores with graph awareness. In: Proceedings of the USENIX symposium annual technical conference, USENIX ATC
- Sankaranarayanan J, Samet H, Teitler BE, Lieberman MD, Sperling J (2009) TwitterStand: news in Tweets. In: SIGSPATIAL
- Sarwat M, Sun Y (2017) Answering location-aware graph reachability queries on geosocial data. In: Proceedings of the international conference on data engineering, ICDE
- Sarwat M, Elnikety S, He Y, Kliot G (2012) Horton: online query execution engine for large distributed graphs. In: Proceedings of the international conference on data engineering, ICDE
- Sarwat M, Eldawy A, Mokbel MF, Riedl J (2013a) PLUTUS: leveraging location-based social networks to recommend potential customers to venues. In: Proceedings of the international conference on mobile data management, MDM
- Sarwat M, Elnikety S, He Y, Mokbel MF (2013b) Horton+: a distributed system for processing declarative reachability queries over partitioned graphs. PVLDB 6(14):1918–1929
- Sarwat M, Levandoski JJ, Eldawy A, Mokbel MF (2014) LARS*: an efficient and scalable location-aware recommender system. IEEE Trans Knowl Data Eng TKDE 26(6):1384–1399
- Sarwat M, Moraffah R, Mokbel MF, Avery JL (2017) Database system support for personalized recommendation applications. In: Proceedings of the international conference on data engineering, ICDE
- Shao B, Wang H, Li Y (2013) Trinity: a distributed graph engine on a memory cloud. In: Proceedings of the ACM international conference on management of data, SIGMOD
- Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: ICDE, pp 148–159
- Tweet Tracker (2013) TweetTracker: track, analyze, and understand activity on Twitter. <http://tweattracker.fulton.asu.edu/>
- Watanabe K, Ochi M, Okabe M, Onai R (2011) Jasmine: a real-time local-event detection system based on geolocation information propagated to microblogs. In: CIKM
- Wu L, Lin W, Xiao X, Xu Y (2013) LSII: an indexing structure for exact real-time search on microblogs. In: ICDE
- Yao J, Cui B, Xue Z, Liu Q (2012) Provenance-based indexing support in micro-blog platforms. In: ICDE

Spatiotemporal Data Integration

► Spatial Data Integration

Spatiotemporal Data: Trajectories

Xiaofang Zhou and Lei Li

School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD, Australia

Synonyms

Moving objects; Routes; Spatiotemporal representation; Traces

Definitions

Let $p(l, t)$ be a *spatiotemporal point* with location l at time t . A *trajectory* is defined as $\tau = < p_1, p_2 \dots p_n >$ where $p_i.t \leq p_j.t$ if $i < j$. That is, a trajectory is a sequence of spatiotemporal points ordered by time.

Location l can be represented as a longitude and latitude pair in geographical space or a road segment ID and distance offset in a road network. A trajectory without temporal information is often called *route* or *path*, and a collection of

trajectories of an object is called its *trace*. The trajectory with a specific origin and destination pair (OD pair) is also called a *trip*.

Overview

A trajectory records how an object moved in a space. Such information is easier than ever to acquire with the prevalence of location-capturing devices such as GPS nowadays. Therefore, large volumes of trajectory data are being accumulated from various sources every day, for animals, human, vehicles, and natural phenomena (Zheng 2015). Animal trajectory data can be obtained by attaching tracking devices to animals, for environment protection and animal behavior studies. *Movebank* has collected animal movement data from thousands of studies at millions of locations. Human trajectories are collected from travelers, cyclists, and joggers, due to the recent popularity of electronic fitness tracking devices and mobile devices. Transport-related trajectory data, which by far are the most voluminous, most interesting, and most useful type of trajectory data, are generated by GPS devices and fixed-location data-capturing devices from vehicles, airplanes, and ships. Taxi service providers like *Uber* and *DiDi* create terabytes of trajectory data every single day. Natural phenomena trajectory data are also collected for scientific studies. *NOAA Air Resources Laboratory* (Draxler and Rolph 2003) stores a massive amount of meteorology trajectories that can be used to better understand the causes and impacts of natural disasters and to protect the natural environment.

The works on trajectory data can be categorized into several topics (Zheng and Zhou 2011). The first one is *trajectory preprocessing*, including noise removal to improve data quality, map matching that aligns points to road segments for road network-constrained moving objects (such as cars), data compression, and trip segmentation that prepares data for further uses like clustering and classification. The second one is related to *trajectory data management*, which aims at answering retrieval queries efficiently by building indexes and developing query algorithms. The

third one is *trajectory mining*, which involves finding the patterns among the trajectories, classifying them into different categories, detecting outliers, and reducing the uncertainty between two consecutive points. Finally, based on all the previous steps, trajectory data can be used to solve problems ranging from more conventional applications such as traffic condition prediction and route planning to the more recent applications such as fuel and pollution emission minimization in a city.

Key Research Findings

Trajectory Preprocessing

Essentially, the raw trajectory is an array of (l, t) data, which can be noisy, too dense, or too coarse in terms of sampling rates and cannot be directly used for a variety of applications. Therefore, like any other types of raw data, preprocessing is needed before actual uses.

Noise Removal

Due to the accuracy of the devices, the data collected are not always accurate. Some of the data points obviously drift off the course. The simplest approach is using the mean or median value of a sliding window to filter out the noise point. However, it fails when there are multiple consecutive noise points. More practical approaches apply outlier detection methods, like computing the travel speeds of the points and removing those that surpass the threshold (Yuan et al. 2013).

Map Matching

If there exists an underlying road network that confines object movement (e.g., for cars), it is always beneficial to attach the GPS points to the corresponding roads. Based on the time when the matching is executed, it can be categorized into *real-time mode* and *post-processing mode*. The real-time map matching is widely applied in real-time turn-by-turn navigation systems. It requires fast computation and can only use the previous few points (i.e., no future points can be used), while it cannot guarantee the continuity on the path during the trip. The post-processing map

matching can utilize the entire trajectory, so it is more accurate but time-consuming.

The techniques used in map-matching methods can be divided into four groups. The first one mainly considers the geometry distance between GPS point/trajectory segment and the candidate map points/map edges that could be possibly aligned on. The second group also considers topology of a map such as connectivity and contiguity of roads. The third group further improves the accuracy by using probability-based methods like the *Hidden Markov Model* and *Kalman Filter*. More advanced approaches combine the existing methods with additional information like Wi-Fi, Bluetooth, and cellular fingerprint on mobile phones, driver behaviors, and other semantic information about the road network, the objects, and other related information.

Compression

The amount of trajectory data increases at an increasing pace, leading to gigantic storage overhead as well as computation and communication costs. However, not many applications need the trajectory data to be that precise, so compression is necessary in many cases.

A simple approach to reduce the size is to remove some points if they do not affect the precision dramatically. In this way, the new and more compact trajectory is an approximation of the original one. Another approach takes advantages of the road network if applicable. Data size can be reduced significantly after using consecutive matched road segments to represent points because normally there are multiple GPS points along the same road segment. Further compression can be achieved with the help of frequent sequential pattern mining and Huffman Coding (Song et al. 2014), or using other string compression methods like *Burrows-Wheeler Transform*, because a series of roads can be viewed as a string with each road representing a character in the alphabet (Koide et al. 2017).

Segmentation

In many high-level applications where trajectory data are used (such as traffic and traveler behavior analytics), shorter trajectory segments

make more sense than the original long trajectory. This is not only because shorter trajectories can better support similarity-based analysis but also improve computation efficiency (many trajectory similarity measures are of quadratic complexity). Further, segmentation based on OD pairs can also bring semantic information to trajectories. Trajectory segmentation (or trip segmentation) is the process that breaks a long trajectory into a series of short trajectories.

The segmentation processing has three main categories. Firstly, the trajectory can be segmented based on time interval. It is like resampling the original trajectory on a lower sampling rate. Secondly, it can be segmented based on the shape (Lee et al. 2007), which involves finding the turning points. Finally, semantic meaning of the points (like walk segment, driving segment, and segments between taxi waiting time) can also serve as segmentation points.

Trajectory Management

Querying and processing directly on a large volume of trajectories are actually very time-consuming. Therefore, how to organize and index the trajectory data to support trajectory query answering efficiently becomes a research topic, which is called trajectory management (Deng et al. 2011).

Trajectory Query

Based on the type of query entity (i.e., points, regions, and trajectories), trajectory queries can be classified into three types. *P-Query* asks for points which satisfy a given spatiotemporal relationship to specified trajectory segment(s) (e.g., top-k nearest neighbors) or reversely. Similarly, *R-Query* asks for regions, and *T-Query* asks for trajectories. An example of a spatiotemporal relationship is “within 500 m of a gas station between 9:00pm and 9:30pm.”

Trajectory Index

Compared with other general data types, trajectory data has unique characteristics like continuous long time span. Meanwhile, queries on trajectory also often ask for information

in a continuous time window. Based on these characteristics, three types of indexes are proposed.

The first type augments the existing multi-dimensional index with a temporal dimension, like *3D R-Tree*. The second type further breaks the temporal dimension down to multi-version structures, such as *HR⁺-Tree* and *MV3R-Tree* (Tao and Papadias 2001). The third type focuses on dividing the spatial dimension into grids and then building a separate temporal index on each grid. This category includes *SETI* and *MTSB-Tree*.

Trajectory Similarity

Like any other data types, a similarity (or distance) measurement is needed to compare between trajectories.

The simplest scenario is the distance from a point to a trajectory, which is measured by the distance to the nearest point in the trajectory. As for the distance between a set of nodes and a trajectory, the closer matched pair of points is assigned with larger weights using the sum of distance, while those faraway pairs are given much lower value typically in an exponential way.

The similarity between two trajectories is usually measured by some kind of aggregation of distances between trajectory points (Wang et al. 2013). Along this line, several typical similarity functions for different applications include *Closest Pair Distance*, *Sum-of-Pairs Distance*, *Dynamic Time Warping*, *Longest Common Subsequence*, *Edit Distance with Real Penalty*, and *Edit Distance on Real Sequences*. It is worth noting that some of those similarity functions were originally proposed for time series data. But as trajectories can be regarded as a special kind of time series in a multidimensional space, these similarity functions can also be applied to trajectory data.

Trajectory Uncertainty

Because trajectory data is always a sample of the object's actual movement trace, the uncertainty exists between any two points in a trajectory especially when the sampling rate is low (Zheng

et al. 2012). On one hand, some works aim to reduce the uncertainty of the trajectory. On the other hand, other works try to add more uncertainties for privacy protection reasons.

The first group of researches focuses on providing conservative bounds for the positions of uncertain objects between two points, which is achieved by employing geometric cylinders or beads. Independent probability density functions can be used to model the uncertain positions (Cheng et al. 2004).

The other group of approaches aim at providing the most k likely routes between sample points with the help of a set of uncertain trajectories, because these trajectories that share similar routes can often supplement each other to make themselves more complete (Su et al. 2013).

Contrary to the previous attempts, techniques are developed to work on preventing user privacy leaking by blurring the published trajectory while preserving the utility of the data.

Trajectory Mining

Trajectory Pattern Mining

Trajectory pattern mining aims at discovering trajectory groups based on their proximity in either a spatial or a spatiotemporal sense. There are four main categories of patterns that can be discovered from a single trajectory or a group of trajectories.

The first one is the *moving together pattern*, which discovers a group of objects that move together for a certain time period. A *flock* is a group of objects that travel together within a disk of some user-specified size for at least k consecutive timestamps. Apparently, the fixed disk shape with a fixed size can be too strict and rigid to use in practice, so *convoy* is proposed by finding patterns based on density. In this way, patterns of any shape and size can be discovered. However, both flocks and convoys are strict on period, so *swarm* (Li et al. 2010a) is proposed to further generalize the cluster with objects lasting for at least k timestamps. To cope with stream data, *traveling companion* uses a data structure (called traveling buddy) to continuously find convoy-/swarm-like patterns from trajectories and can work online.

By allowing the membership of a group to evolve gradually, *gathering* (Zheng et al. 2014) can be used to detect events and incidents.

The second one is *trajectory clustering*. Unlike general clustering tasks that use feature vectors to represent objects, it is hard to generate a uniform feature vector because different trajectories contain different and complex properties, such as length, shape, sampling rate, number of points, and their orders. A number of works have been done using the trajectory similarity. Although some of them work on the entire trajectory, it is rare for two objects traveling together for the entire journey. So more practical approaches partition trajectories into segments before clustering. If the trajectories are matched to map, the trajectory clustering task can be done by applying graph clustering algorithms.

The third one is *mining sequential patterns from trajectories*. A *sequential pattern* means a certain number of moving objects travel a common sequence of locations in a similar time interval and the locations of the sequence do not have to be consecutive. A general solution is using trajectory clustering first and then reforming trajectories with cluster IDs. In this way, existing sequential pattern mining algorithms like *PrefixSpan* can be used. If the trajectory can be matched on map, the resulting sequence of road IDs can use *Suffix Tree* to find the frequent patterns (Song et al. 2014).

The last one is *mining periodical patterns from trajectories*. Some object movements have periodical patterns over the long history. For example, people go to work in the morning and go back home at night. Animals migrate from one place to another at different time of the year. A straightforward approach is to use general frequent pattern mining methods. However, real-life periodic behaviors are complicated and involve multiple interleaving periods, partial time span, and spatiotemporal noises and outliers. Therefore, a more advanced two-stage method is proposed (Li et al. 2010b). In the first stage, it mines all the frequent visiting places by density-based clustering algorithms. The temporal data corresponding to entering and leaving these places can be used to find the period values. In the second stage,

larger periodic patterns are created by applying hierarchical clustering algorithms on the partial movement sequences.

Trajectory Classification

Trajectory classification helps divide trajectories into different statuses. For example, taxi trajectories can be *occupied*, *non-occupied*, and *parking*. A cell phone user can be *stationary*, *walking*, and *driving* or even *driving*, *biking*, *commuting by bus*, and *walking*. In general, the classification has three steps. First of all, trajectories are divided into segments in preprocessing stage. After that, features of each segment are extracted. Finally, existing sequence inference models (such as *Dynamic Bayesian Network*, *Hidden Markov Model*, and *Conditional Random Field*) can be used.

Outliers Detection

Outliers of trajectory data can be points significantly different from others spatially or temporally and can also be observations that do not follow the expected patterns or constraints. One general approach is to leverage standard frequent pattern mining methods. If the trajectory cannot fall into any cluster, it might be an outlier (Lee et al. 2007).

Examples of Application

Trajectory data can be found in many applications; here we just list a few as examples.

Travel Recommendation. It aims to find interesting locations and travel sequences from trajectories generated by many people. (Zheng and Xie 2011) identifies staying points from users' trajectories and clusters these points into locations of interest. After that, it can identify the top-k most interesting locations and travel experts in a city and do the recommendations based on their data. Moreover, it can recommend trajectories themselves, because historical traveling experiences can also reveal valuable information on how other people usually choose routes between locations.

Traffic Condition Estimation. The trajectories of vehicles on the road can reflect the traffic condition (Yang et al. 2013). It needs a series of

processing to generate a speed profile from trajectories: map matching, speed generation, missing value estimation, and compression. The result not only can be used for finding the fastest path from one place to another at different departure time but also can help find the congestion of road network and provide decision support for urban planning.

Map Inference. Normally, vehicle trajectories can always be matched to some roads on a map. However, when a new road is developed and the map has not been updated, or if there is even no map for the current region, map matching will fail. Map inference works under this scenario to infer new maps or update existing maps based on trajectories.

Diagnosing Traffic Anomalies. Such examples can be that a taxi driver takes a malicious detour, that an unexpected road change occurs, or that people travel a wrong path. They can all be discovered by trajectory outlier detection using trajectory clustering.

Future Movement Prediction. Periodic trajectory pattern mining can be used to predict the next direction or destination of the current moving objects, like a group of animals or a commuter. The prediction can further help compress the trajectory data itself.

User Similarity Estimation. There are many aspects of similarity between users, like connections from social network, point of interest, check-in history, and any other logs, that can be obtained. Besides them, trajectory data, which reveals the life patterns of the users by trajectory classification and clustering, can also help improve the accuracy of similarity comparison.

Sport Tactic Analysis. For many team games like soccer, basketball, hockey, and rugby, the players' movements are essentially trajectories. By analyzing the video data from different cameras, the trajectory of each player can be reconstructed and are used in tactic analysis by many professional clubs nowadays. Some of them even hire a data analyst as a coaching staff.

Airspace Monitoring and Aircraft Guiding. A large volume of aircraft trajectory data is managed by an air traffic controller. They use

these trajectories to monitor the “health” of the airspace, which is implemented by trajectory clustering and outlier detection.

Scientific Study. Meteorologists use trajectory of SO_2 and NO_x in an isentropic and constant level to analyze the contributions of acidic deposition. Zoologists use the trajectory of animals to study their movement patterns. Biologists use proteomic trajectories to study mouse retina development.

Future Directions for Research

As summarized above, there exists a rich body of research on all aspects of trajectories, from data capturing, cleaning, compression, and indexing to processing. We have witnessed an accelerating trend of research activities from both academic and industry on this topic. There are three main drivers in today's big data landscape, which will also drive the research in trajectory data management, processing, and analytics in the foreseeable future.

First, it is about volume. Not only the volume of trajectory data now can reach TB level daily for some large navigation or taxi-/car-sharing companies, but also the number of queries has increased dramatically. What is the best way to process map matching for one billion points every day? How can we reduce the processing costs if we have 10,000 shortest path queries in the same region? This is a scenario that exists already, as when a user opens a map-based app and inputs a location, it issues a shortest path query. For every problem we solved before, it is the time to revisit them, to see how they can become scalable using new computing platforms and how better algorithms can be designed to support batch query processing (for a very large number of streaming queries on streaming data).

Second, it is about semantics. After all, trajectory data are low-level data which can be noisy and with a high level of redundancy (among consecutive points of one moving object, among the history data of one moving object over a long period, and among objects with similar moving patterns). There is a dilemma between justifying

the high costs of storing all data available and the fear that some data which we only find useful in the future for some purposes we do not know yet might be lost if we do not store them. Clearly, a new way of thinking is needed to manage trajectory data based on a semantic hierarchy, from raw data, calibrated data, events detected, summarization of data for a basic set of requirements, patterns discovered, and general statistics. Data can be gradually reduced over time and eventually removed. In such a way, trajectory data can be at the center for data integration and data analytics, as location and time are two ubiquitous dimensions for most information useful to us. Trajectory will no longer be considered as specialized data with limited applications; rather, it is an enabler data asset underpinning the future of a data-rich society.

Third, trajectory data can reveal so much about a person. With so much location and movement data about a person captured and accessed so easily, security and privacy become an extremely serious issue for trajectory data. Simple facts about one visiting or not visiting a place can be highly sensitive. This problem can become much more significant when the trajectories are used with other data sources including social network data. Some research has already started on this topic, but much more is needed urgently.

Cross-References

- ▶ [Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems](#)
- ▶ [Big Data in Smart Cities](#)
- ▶ [Data Cleaning](#)
- ▶ [Indexing](#)
- ▶ [Spatial Data Mining](#)
- ▶ [Storage Technologies for Big Data](#)

References

- Cheng R, Kalashnikov, DV, Prabhakar, S (2004) Querying imprecise data in moving object environments. *IEEE Trans Knowl Data Eng* 16(9):1112–1127
- Deng K, Xie K, Zheng K, Zhou X (2011) Trajectory indexing and retrieval. *Computing with spatial trajectories*. Springer, New York, pp 35–60
- Draxler RR, Rolph, GD (2003) Hysplit (hybrid single-particle lagrangian integrated trajectory). NOAA air resources laboratory, silver spring, MD. model access via NOAA ARL ready website
- Koide S, Tadokoro Y, Xiao C, Ishikawa Y (2017) CiNCT: compression and retrieval for massive vehicular trajectories via relative movement labeling. *arXiv preprint arXiv:1706.02885*
- Lee J-G, Han J, Whang K-Y (2007) Trajectory clustering: a partition-and-group framework. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, pp 593–604
- Li Z, Ding B, Han J, Kays R (2010a) Swarm: mining relaxed temporal moving object clusters. *Proc VLDB Endow* 3(1–2):723–734
- Li Z, Ding B, Han J, Kays R, Nye P (2010b) Mining periodic behaviors for moving objects. In: *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp 1099–1108
- Song R, Sun W, Zheng B, Zheng Y (2014) Press: a novel framework of trajectory compression in road networks. *Proc VLDB Endow* 7(9):661–672
- Su H, Zheng K, Wang H, Huang J, Zhou X (2013) Calibrating trajectory data for similarity-based analysis. In: *Proceedings of the 2013 ACM SIGMOD international conference on management of data*. ACM, pp 833–844
- Tao Y, Papadias D (2001) The mv3r-tree: a spatio-temporal access method for timestamp and interval queries. In: *Proceedings of very large data bases conference (VLDB)*, 11–14 Sept, Rome
- Wang H, Su H, Zheng K, Sadiq S, Zhou X (2013) An effectiveness study on trajectory similarity measures. In: *Proceedings of the twenty-fourth Australasian database conference*, vol 137. Australian Computer Society, Inc., pp 13–22
- Yang B, Guo C, Jensen CS (2013) Travel cost inference from sparse, spatio temporally correlated time series using Markov models. *Proc VLDB Endow* 6(9): 769–780
- Yuan J, Zheng Y, Xie X, Sun G (2013) T-drive: enhancing driving directions with taxi drivers' intelligence. *IEEE Trans Knowl Data Eng* 25(1):220–232
- Zheng Y (2015) Trajectory data mining: an overview. *ACM Trans Intell Syst Technol (TIST)* 6(3):29
- Zheng Y, Xie X (2011) Learning travel recommendations from user-generated GPS traces. *ACM Trans Intell Syst Technol (TIST)* 2(1):2
- Zheng Y, Zhou X (2011) Computing with spatial trajectories. Springer Science & Business Media, New York
- Zheng K, Zheng Y, Xie X, Zhou X (2012) Reducing uncertainty of low-sampling-rate trajectories. In: *2012 IEEE 28th international conference on data engineering (ICDE)*. IEEE, pp 1144–1155
- Zheng K, Zheng Y, Yuan NJ, Shang S, Zhou X (2014) Online discovery of gathering patterns over trajectories. *IEEE Trans Knowl Data Eng* 26(8):1974–1988

Spatiotemporal Graphs Big Data

- ▶ Spatial Graph Big Data

Spatiotemporal Networks Big Data

- ▶ Spatial Graph Big Data

Spatiotemporal Representation

- ▶ Spatiotemporal Data: Trajectories

Spatio-textual Data

Gao Cong¹ and Christian S. Jensen²

¹School of Computer Science and Engineering,
Nanyang Technological University, Singapore,
Singapore

²Department of Computer Science, Aalborg
University, Aalborg, Denmark

Synonyms

Geo-textual data; Spatial and textual data

Definitions

With the proliferation of GPS-equipped mobile devices, notably smartphones, massive volumes of geo-located, or geo-tagged, text content are becoming available. For example, Foursquare hosts over 105 million locations around the world with over 12 billion check-ins (<https://foursquare.com/about> accessed January 2018), where each location is associated with both a geographical location and text content and similarly each check-in

is also associated with a location and text. Facebook also supports location check-ins. We refer to such data as *geo-textual*, or *spatio-textual*, data. Other examples of such data include points of interest (POIs) with descriptive text, geo-tagged microblog posts (e.g., tweets), geo-tagged photos with text tags (e.g., as found at Flickr and Instagram), geo-tagged news, and geo-tagged web pages. Spatio-textual data can be divided into (i) streaming spatio-textual data that arrives at a high rate, exemplified by geo-tagged tweets, and (ii) static spatio-textual data that is relatively stable, exemplified by collections of POIs.

Overview

Massive volumes of spatio-textual data are available from many sources. Furthermore, as new spatio-textual data is being generated, the volumes will continue to grow at rapid pace as mobile devices continue to proliferate. Additionally, spatio-textual data often comes with rich context information, such as temporal information. With this development as the backdrop, many research problems and solutions have been proposed for managing, analyzing, and mining spatio-textual data. We proceed to cover three representative types of research problems.

Spatial keyword queries To support the querying or search of spatio-textual data, many types of spatial keyword queries have been proposed. In general terms, a spatial keyword query takes a location and keywords and arguments and finds the spatio-textual objects that best match the location and keywords. For example, a user may want to find a nearby place whose textual description is relevant to “fine arts.” A number of index structures and query processing algorithms (Zhou et al., 2005; Hariharan et al., 2007; Felipe et al., 2008; Cong et al., 2009, 2012; Wu et al., 2012a,b; Rocha-Junior et al., 2011; Khodaei et al., 2010; Vaid and Jones, 2005; Zhang et al., 2013a,b; Chen et al., 2006; Christoforaki et al., 2011) have been developed for efficiently processing different kinds of spatial keyword queries.

Querying spatio-textual streams Spatio-textual data may arrive at a high rate (e.g., geotagged tweets, photos, or check-ins) and can then be modeled as data streams. In such streaming settings, continuous queries are of particular interest as users may want to be notified when interesting spatio-textual objects arrive. Several solutions (Chen et al., 2013, 2015; Li et al., 2013; Wang et al., 2015; Hu et al., 2015) for continuously querying spatio-textual streams have been proposed. For example, a user may want to be notified when tweets arrive that contain the term “flu” and are posted from within 5 km of the user’s home.

Exploring spatio-textual data When users do not have clear ideas about what to query or search for, it is beneficial to provide functionality that enables users to search, navigate, and discover new facts from spatio-textual data. Several proposals exist on exploring spatio-textual data (Feng et al., 2016; Skovsgaard et al., 2014; Zhao et al., 2016). For example, when a user is exploring a region on a map, the user can be shown the top- k most frequent terms in the region, and the result can be updated interactively when the user slides over the map.

Key Research Findings

Spatial keyword queries Consider a set \mathcal{D} of spatio-textual objects. An object $p \in \mathcal{D}$ is a two-tuple: $\langle \lambda, \psi \rangle$, where λ encodes a geo-location and ψ is a text value. Many types of spatial keyword queries on spatio-textual data exist that target different applications. The most standard spatial keyword queries generalize fundamental queries from spatial databases and information retrieval. In spatial databases, the arguably most fundamental queries are range and k nearest neighbor queries. In information retrieval, queries may be Boolean keyword expressions, returning results that satisfy the Boolean expressions, or ranking-based, returning the k objects that are most similar to query keywords according to

some text similarity function. Basic spatial keyword queries return a set or ranked list of objects from \mathcal{D} .

Four types of basic queries are covered below. Let ρ be a spatial region, λ be a point location, τ be a Boolean keyword expression, ψ be a set of keywords, and k be the number of objects to return. (1) A *Boolean range query* $q = \langle \rho, \tau \rangle$ returns all objects in \mathcal{D} that are located in region ρ and that satisfy the Boolean expression τ . An example is “Retrieve all objects whose text description contains the keywords *vegetarian*, *pizza*, and *latte* and whose location is within 1 km of the query location.” (2) A *Boolean kNN query* $q = \langle \lambda, \tau, k \rangle$ returns up to k objects from \mathcal{D} , each of which satisfies the Boolean expression τ , ranked in increasing spatial distance from λ . An example is “Retrieve the k objects nearest to the query location such that each object’s text description contains the keywords *vegetarian*, *pizza*, and *latte*.” (3) A *top- k range query* $q = \langle \rho, \psi, k \rangle$ returns up to k objects from \mathcal{D} that are located in the query region ρ , now ranked according to their text relevance to the set of keywords ψ . Here, an example is “Retrieve up to k objects whose locations are within 1 km of the query location, and that have the highest ranking scores, measured by the relevance of their text descriptions to the query keywords *vegetarian*, *pizza*, and *latte*.” (4) A *top- k kNN query* $q = \langle \lambda, \psi, k \rangle$ retrieves k objects from \mathcal{D} , ranked according to a function that takes into consideration both spatial proximity and text relevance. This functionality is exemplified by the query “Retrieve the k objects with the highest ranking scores according to a function that combines their distance to the query location (a point) and the relevance of their text description to the query keywords *vegetarian*, *pizza*, and *latte*.”

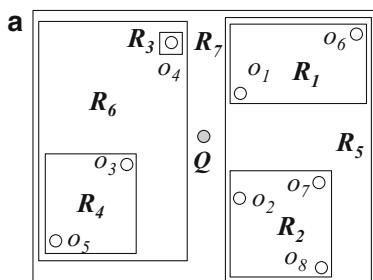
Index structures and algorithms have been developed to enable efficient processing of spatial keyword queries. Existing spatial keyword indexing techniques often combine a spatial index and a text index so that both textual and spatial information can be utilized to prune the search space when processing spatial keyword queries. The existing indices can be categorized according

to the spatial index they utilize: (i) R-tree-based indices (Zhou et al., 2005; Hariharan et al., 2007; Felipe et al., 2008; Cong et al., 2009; Wu et al., 2012a,b; Rocha-Junior et al., 2011), (ii) grid- or quad-tree-based indices (Khodaei et al., 2010; Vaid and Jones, 2005; Cong et al., 2012; Zhang et al., 2013a,b), and (iii) space-filling curve-based indices (Chen et al., 2006; Christoforaki et al., 2011).

The grid-based indices combine a spatial grid index and an inverted index for handling spatio-textual data, with two obvious ways of combining the two types of indices being the spatial primary index and the text primary index (Vaid and Jones, 2005). In the spatial primary index, spatio-textual objects are first organized by a grid index based on their spatial information. Then, for each grid cell, an inverted file is built for the objects that fall in the cell. In contrast, the text primary index extends the inverted index with the spatial information. Specifically, in the text primary index, each word is associated with a postings list, in which postings, each representing id of an object containing the word, are organized by a grid cell based on their locations. Each posting in a postings list contains the id of an object that contains the word of the postings list, and the postings are organized in a grid according to the locations of their objects. Using the example set of spatio-textual objects shown in Fig. 1a, b, the structure of the text primary index is exemplified in Fig. 2. The two index struc-

tures are proposed for tackling the problem of retrieving web documents relevant to a keyword query within a prespecified spatial region, i.e., the Boolean range query. For example, using the spatial primary index, at query time a set of cells that intersects with the query region is retrieved first. Then, the objects in these cells whose documents satisfy the query keywords are returned as the result. There exists no method to extend the two indices to efficiently handle the other three types of basic queries. A straightforward extension would not work better than using only an inverted file.

The R-tree-based geo-textual indices typically combine the R-tree index and the inverted index. The IR-tree (Cong et al., 2009) is a representative structure in this category. The index augments each node of the R-tree with a summary of the text content of the objects under the node. Specifically, each node contains a pointer to an inverted file that summarizes the text content of the objects in the subtree rooted at the node. The inverted file for a node contains (1) a vocabulary of all distinct terms in the text descriptions of the objects in the node's subtree and (2) each term t that is associated with a postings list, which is a sequence of pairs $\langle c, wt_{cp,t} \rangle$. Here, if the node is a non-leaf node, c is a child node of the node, and $wt_{c,t}$ is the number of times term t occurs in the object in the subtree that contains t the most times. If the node is a leaf node, c is an object in the node, and $wt_{c,t}$ is the frequency of



Spatio-textual objects and their bounding rectangles

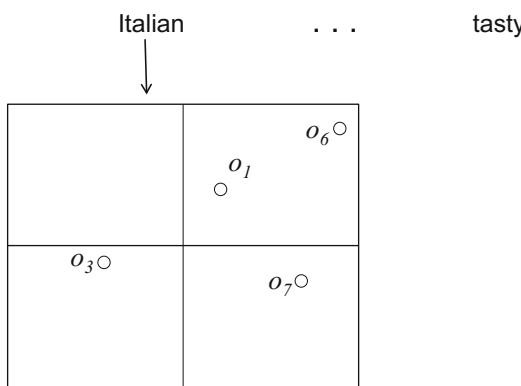
b	object	terms and term frequencies
o_1	(Italian, 2)(restaurant, 2)(pasta, 1)	
o_2	(coffee, 3)(restaurant, 3)(tasty, 1)	
o_3	(Italian, 1)(pizza, 1)(tasty, 1)	
o_4	(restaurant, 1)(pizza, 1)(tasty, 1)	
o_5	(coffee, 4)(restaurant, 3)	
o_6	(Italian, 4)(restaurant, 4)	
o_7	(Italian, 1)(coffee, 1)(restaurant, 4)(pasta, 1)	
o_8	(coffee, 3)(restaurant, 3)(tasty, 1)	

Text information of objects in (a)

Spatio-textual Data, Fig. 1 Example. (a) Spatio-textual objects and their bounding rectangles. (b) Text information of objects in (a)

term t in the object. The structure of an IR-tree is illustrated in Fig. 3.

The IR-tree supports all the four types of basic queries. We proceed to use the Boolean range query and top- k k NN query to illustrate how the IR-tree can be used to handle the basic queries. The high-level algorithm for using the IR-tree to handle the Boolean range query starts at the root node of an IR-tree and checks each entry in the node. If an entry overlaps the query

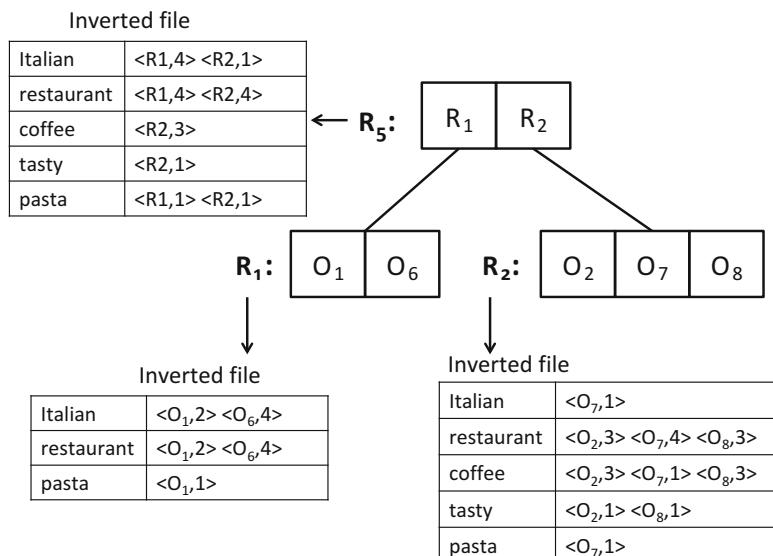


Spatio-textual Data, Fig. 2 Grid structure for keyword *Italian*

region and satisfies the keyword expression, the algorithm will check the entry's child node. The checking is done recursively over the IR-tree. When a leaf node is reached, the spatio-textual objects in the leaf nodes are checked, and objects satisfying both the spatial condition and the Boolean keyword condition are returned as results. Note that each non-leaf node contains summary information on both the spatial location and the terms of the objects in the node's subtree. For example, in node R_5 in Fig. 3 tells which terms the inverted file associated with the node will tell what keywords are contained in the objects in the node's subtree. If a query targets objects containing the term “rice,” the inverted file tells that there is no need to search R_5 's subtree.

The key idea of employing the IR-tree to compute the top- k k NN query is that each tree node on the summary information for each entry can be used to estimate a good upper bound on the relevance score, considering both spatial proximity and keyword relevance, to the query. This makes it possible to extend the best-first traversal algorithm, a standard algorithm for efficiently computing spatial k NN queries using the R-tree, to compute the top- k k NN query. The algorithm

**Spatio-textual Data,
Fig. 3** IR-tree and its inverted files



starts at the root and maintains a priority queue to keep track of the nodes and objects that have yet to be visited, for which the estimated upper bounds for each node, or the spatio-textual relevance score for each object, are used as the keys of nodes or objects, respectively. When deciding which node to visit next, the algorithm picks the node with the best key value from the set of nodes that have yet to be visited. The algorithm stops when the upper bounds of the remaining nodes are smaller than the currently k best results. The benefit of using the IR-tree for computing spatial keyword queries is that both spatial and textual information can be used in a combined fashion to prune the search space during query processing.

There also exist spatio-textual indices that combine a space-filling curve, e.g., the Z-order curve, and the inverted file. A typical example is the SFC-QUAD index (Christoforaki et al., 2011), which extends the inverted file with spatial information using space-filling curve. Specifically, the docIDs in each inverted list are assigned and ordered based on their spatial positions along a Z-order curve. SFC-QUAD is designed for *Boolean range query*. When processing a query with a given spatial region, a set of object ID ranges that fall in the query region are found. Then, these ranges are merged into a smaller number of ranges for reducing random disk I/O costs. Subsequently, the corresponding parts of the inverted lists of the query keywords within the ID ranges are retrieved, and then existing techniques for checking the Boolean expression of query keywords can be employed on the retrieved inverted lists. There exists no method to extend this type of index to efficiently handle the other three types of basic queries.

Beyond the basic spatio-textual queries, many other types of spatio-textual queries exist that are designed for different types of user needs. For example, the *m-closest keywords* (*mCK*) query (Zhang et al., 2009, 2010; Guo et al., 2015) retrieves a set of objects whose text content together contains a set of m query keywords and such that the maximum distance between any two objects is minimized. Another example is the *collective keyword* query (Cao et al., 2011, 2015; Long

et al., 2013), which takes a location λ and a set of keywords ψ as arguments. Its search space is all subsets of the set of objects \mathcal{D} , and it returns a set of objects such that (i) the textual descriptions of these objects collectively cover ψ , (ii) the result objects are all close to λ , and (iii) the result objects are close to each other. These problems of retrieving a group of objects satisfying keyword covering condition while minimizing the distance within the group or between the group and query are NP-hard, and heuristic and approximation solutions are developed to answer these queries efficiently.

Querying streaming spatio-textual data On streaming spatio-textual data, the four kinds of spatial keyword queries can be posed as snapshot queries. Furthermore, subscription queries on streaming spatio-textual data are often of great interest. A typical type of subscription query on spatio-textual data streams is the *Boolean subscription query* (Chen et al., 2013; Li et al., 2013; Wang et al., 2015), in which both spatial and keyword conditions serve as Boolean filters. For example, a user may want to subscribe to tweets that are posted within 500 m of the user's office and that contain the terms "sales" and "clothing," to receive such tweets continuously.

Other kinds of subscription queries also exist. Specifically, top- k subscription queries return a subscriber only the top- k results, which are ranked by considering text relevance, spatial proximity, and the freshness of the object with respect to the query. For example, a subscription query may be submitted for a POI (e.g., a hotel) over streaming geo-tagged tweets, where the location of the POI is the query location and selected keywords from the POI's text description (e.g., service, cleanliness, free Wi-Fi) can be used as the query keywords. Such a subscription query may be of interest to a management team of a POI that wants to be continuously fed with the top- k tweets, ranked based on the their relevance to the query keywords, their proximity of tweets to the query location, and their freshness.

Each user may submit multiple subscription queries, and the number of Internet users is huge. Hence, the number of subscription queries may

be large. Thus focus has been on developing efficient solutions to handle large numbers of spatial keyword subscription queries over spatio-textual streams (Chen et al., 2013; Li et al., 2013; Mahmood et al., 2015; Wang et al., 2015; Chen et al., 2015).

The high-level idea of existing solutions is to group queries such that the queries in one group can be processed together over the spatio-textual data stream, rather than being processed individually, which is much more computationally expensive. Intuitively, similar queries should be grouped together so that good filtering conditions for a group can be designed to check whether a new spatio-textual object can be a result for some queries in the group. In addition to being able to efficiently process a large number of subscription queries over spatio-textual data streams where spatio-textual objects arrive at a high rate, such solutions must be capable of efficiently handling the arrival of new subscriptions and the discontinuation of existing subscriptions. The existing solutions also involve index structures to organize subscription queries for efficient query processing.

Exploring spatio-textual data There exist at least two types of research for exploring spatio-textual data. First, the region search problem aims to identify a region of a user-specified shape and size that maximizes or minimizes some aggregate score of the objects inside it (e.g., the sum of relevance of objects to a query), for user exploration. For certain aggregation scores like SUM, algorithms (Imai and Asano, 1983; Nandy and Bhattacharya, 1995; Choi et al., 2012) with complexity $O(n \log n)$ have been proposed, where n is the number of objects in \mathcal{D} . For more general aggregation function, such as submodular monotone score function, a solution with the worst-case complexity $O(n^2)$ exists (Feng et al., 2016). To improve the efficiency, an approximate algorithm is proposed to select a set T of spatial points from the space for representing spatial objects in \mathcal{D} . The selected points together preserve some properties of \mathcal{D} such that the result region found on the set of points T can be an approx-

imation of the result on \mathcal{D} with performance guarantees (Feng et al., 2016).

Second, rather than finding a region that satisfies a user's needs, the problem of region exploration aims to explore and discover properties of user-specified regions. Given a user-specified region, one study (Skovsgaard et al., 2014) considers the problem of retrieving the top- k frequent words over the geo-textual data stream for the region. A new indexing technique is proposed for counting frequent items in static-sized summaries to allow the summaries to grow and shrink dynamically to adapt to changes in the incoming data. At query time, relevant summaries are merged to provide answers of the top- k frequent words with correctness guarantees.

Another example study of region exploration (Zhao et al., 2016) aims to efficiently discover topics in the spatio-textual data in a user-specified query region. Instead of learning a topic model for each query region, which is time-consuming, a two-phase approach to the exploratory topic mining task is advanced: (1) Indexing and Pre-Computation To support efficient online learning, the geographical space is partitioned into cells, and a grid-based index is constructed to organize the spatio-textual data. Then, by considering the memory constraint and accuracy guarantee of the online topic learning algorithm in the second phase, some cells are selected for learning latent Dirichlet allocation (LDA) models. (2) Online Topic Learning For a given query region, the pretrained topic models on the cells that overlap with the query region are combined as approximate topics for the spatio-textual data in the query region.

S

Examples of Application

Querying, mining, and exploring spatio-textual data have many applications. The basic types of spatial keyword queries are being supported in many online services that host large amounts of spatio-textual data, such as geo-tagged web pages, points of interest, geo-tagged tweets, and check-in. Other types of spatial keyword queries are designed to serve users for scenarios that

the basic queries are not sufficient. For example, consider a query with keywords “hotel, sports shop, and beach.” Perhaps no nearby single place is a good match for all the keywords. Instead, a group of places that are close to each other and are close to the query location may together meet the user’s needs better than any single object. Furthermore, it is expected that more applications will arise as more and more spatio-textual data is being generated in huge amount.

Querying streaming spatio-textual data is also very useful. For example, one application can be annotation of POIs with up-to-date geospatial social updates, such as geo-tagged tweets, as a manager of a POI may be interested in up-to-date tweets whose locations are close to the POI and whose text is relevant to the description of the POI. As another example, Groupon customers register their locations and keywords describing their interests. Then Groupon pushes Groupon messages to those customers that are near the messages and that have keywords that are relevant to the text of the messages.

Exploration of spatio-textual data is a desirable function as the amount of spatio-textual data continues to grow. The region research functionality can be used to find most influential regions for exploration, e.g., finding the best location for a billboard to influence as many consumers as possible to adopt a product, where the consumers may belong to a social network. Another example application of the region search is to find a region with most dense or diverse collection of services and attractions. One application of region exploration is to see what is happening in a region of interest.

Future Directions for Research

First, apart from spatial proximity and text relevance, many factors such as the quality of a spatio-textual object, click-throughs, and diversity can be considered in the ranking of spatial keyword query results. It is natural to consider whether these factors are useful for the ranking of query results and how important they are in ranking results. Reliable evaluation of ranking

functions for spatio-textual data is essential to answer these questions. However, the utility of a result is dependent on the individual user. It is thus challenging to establish a reliable ground truth for the results of ranking queries.

Second, personalized location recommendation (Liu et al., 2017) aims to understand users’ topical interests and mobility preferences from the users’ historical data. Existing work on spatial keyword queries does not consider personalization, while personalized location recommendation does not consider spatial keyword search. It is of interest to attempt to bridge this gap, thus enabling personalized search on spatio-textual data.

Third, it is an open problem to effectively and efficiently support continuous queries on spatio-textual streams. For example, instead of receiving individual tweets from a stream, users may want to be notified in real time of relevant trending events or even of causal relationships among events. Furthermore, high-velocity spatio-textual data streams call for distributed systems to support querying and data analytics.

References

- Cao X, Cong G, Jensen CS, Ooi BC (2011) Collective spatial keyword querying. In: SIGMOD, pp 373–384
- Cao X, Cong G, Guo T, Jensen CS, Ooi BC (2015) Efficient processing of spatial group keyword queries. ACM Trans Database Syst 40(2):13
- Chen Y, Suel T, Markowetz A (2006) Efficient query processing in geographic web search engines. In: SIGMOD, pp 277–288
- Chen L, Cong G, Cao X (2013) An efficient query indexing mechanism for filtering geo-textual data. In: SIGMOD, pp 749–760
- Chen L, Cong G, Cao X, Tan K (2015) Temporal spatial-keyword top-k publish/subscribe. In: ICDE, pp 255–266
- Choi D-W, Chung C-W, Tao Y (2012) A scalable algorithm for maximizing range sum in spatial databases. Proc VLDB Endow 5(11):1088–1099
- Christoforaki M, He J, Dimopoulos C, Markowetz A, Suel T (2011) Text vs. space: efficient geo-search query processing. In: CIKM, pp 423–432
- Cong G, Jensen SC, Wu D (2009) Efficient retrieval of the top-k most relevant spatial web objects. In: PVLDB, pp 337–348
- Cong G, Lu H, Ooi BC, Zhang D, Zhang M (2012) Efficient spatial keyword search in trajectory databases. CoRR, abs/(1205)2880

- Felipe ID, Hristidis V, Rishe N (2008) Keyword search on spatial databases. In: ICDE, pp 656–665
- Feng K, Cong G, Bhowmick SS, Peng W-C, Miao C (2016) Towards best region search for data exploration. In: SIGMOD. ACM
- Guo T, Cao X, Cong G (2015) Efficient algorithms for answering the m-closest keywords query. In: SIGMOD, pp 405–418
- Hariharan R, Hore B, Li C, Mehrotra S (2007) Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In: SSDBM, p 16
- Hu H, Liu Y, Li G, Feng J, Tan K (2015) A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In: ICDE, pp 711–722
- Imai H, Asano T (1983) Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J Algorithms* 4(4):310–323
- Khodaei A, Shahabi C, Li C (2010) Hybrid indexing and seamless ranking of spatial and textual features of web documents. In: DEXA (1), pp 450–466
- Li G, Wang Y, Wang T, Feng J (2013) Location-aware publish/subscribe. In: KDD, pp 802–810
- Liu Y, Pham T, Cong G, Yuan Q (2017) An experimental evaluation of point-of-interest recommendation in location-based social networks. *VLDB* 10(10): 1010–1021
- Long C, Wong RC, Wang K, Fu WA (2013) Collective spatial keyword queries: a distance owner-driven approach. In: SIGMOD, pp 689–700
- Mahmood AR, Aly AM, Qadah T, Rezig EK, Daghstani A, Madkour A, Abdelhamid AS, Hassan MS, Aref WG, Basalamah SM (2015) Tornado: a distributed spatio-textual stream processing system. *VLDB* 8(12):2020–2023
- Nandy SC, Bhattacharya BB (1995) A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Comput Math Appl* 29(8): 45–61
- Rocha-Junior JB, Gkorgkas O, Jonassen S, Nørvåg K (2011) Efficient processing of top-k spatial keyword queries. In: SSTD, pp 205–222
- Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: ICDE, pp 148–159
- Vaid S, Jones CB, Joho H, Sanderson M (2005) Spatio-textual indexing for geographical search on the web. In: SSTD, pp 218–235
- Wang X, Zhang Y, Zhang W, Lin X, Wang W (2015) Ap-tree: efficiently support continuous spatial-keyword queries over stream. In: ICDE, pp 1107–1118
- Wu D, Cong G, Jensen CS (2012a) A framework for efficient spatial web object retrieval. *VLDB J* 21(6): 797–822
- Wu D, Yiu ML, Cong G, Jensen CS (2012b) Joint top-k spatial keyword query processing. *IEEE Trans Knowl Data Eng* 24(10):1889–1903
- Zhang D, Chee YM, Mondal A, Tung AKH, Kitagawa M (2009) Keyword search in spatial databases: towards searching by document. In: ICDE, pp 688–699
- Zhang D, Ooi BC, Tung AKH (2010) Locating mapped resources in web 2.0. In: ICDE, pp 521–532
- Zhang C, Zhang Y, Zhang W, Lin X (2013a) Inverted linear quadtree: efficient top k spatial keyword search. In: ICDE, pp 901–912
- Zhang D, Tan K-L, Tung AKH (2013b) Scalable top-k spatial keyword search. In: EDBT, pp 359–370
- Zhou Y, Xie X, Wang C, Gong Y, Ma W-Y (2005) Hybrid index structures for location-based web search. In: CIKM, pp 155–162
- Zhao K, Chen L, Cong G (2016) Topic exploration in spatio-temporal document collections. In: SIGMOD. ACM

Storage Failure

► Data Longevity and Compatibility

Storage Hierarchies for Big Data

Dirk Duellmann

Information Technology Department, CERN,
Geneva, Switzerland

Motivation

Big data applications usually have to rely on a combination of storage media to achieve an economic balance between the capabilities of different media types and application needs.

Applications requirements vary significantly in data lifetime, number of concurrent data producers/consumers, fraction of active to passive data volume, sharing between parallel processing units, and the relative balance between CPU and IO requirements.

Storage media properties vary by several orders in price per capacity, latency for sequential and random access patterns, aggregate and single stream bandwidth, power requirements, endurance, and reliability. Several methods exist to further adapt these capabilities by combining several storage devices of the same type, but

larger and economically efficient setups are constructed by combining several different storage technologies.

In addition, larger storage deployments typically provide services to more than one application and hence aim to achieve an economic compromise between total cost of ownership and functional capabilities matching the ensemble of their applications.

Tiered Storage and HSM Systems

The above trade-off has since the mainframe days led to storage systems covering several layers in the traditional memory hierarchy (see Fig. 1) and integrate multiple technologies (e.g., disk and tape, disks of different capabilities) into tiered storage systems.

In addition to combining technology capabilities, these systems usually provide an abstraction for accessing data independent of the storage media and mechanisms allowing to move data between the different media.

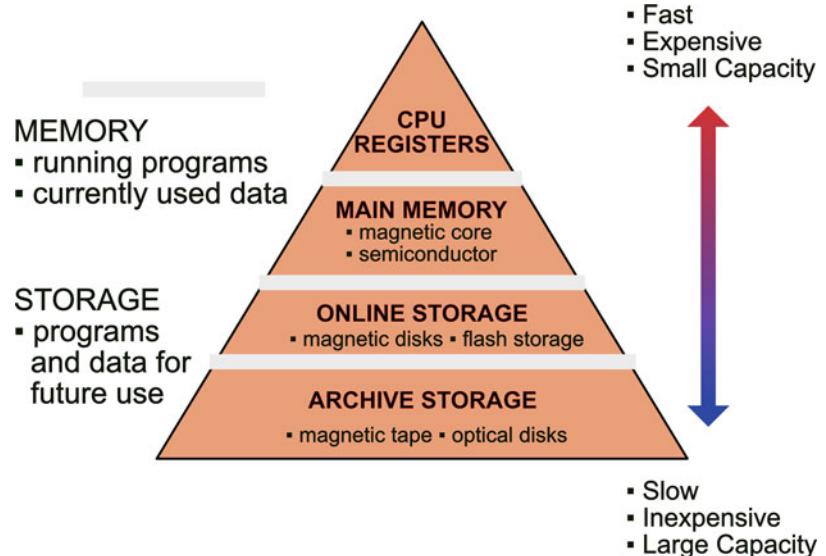
Hierarchical storage management (HSM) systems extend the above abstraction and automatize the data movement between different media or quality of service based on the measured or predicted access patterns.

The resulting storage systems facilitate also the use of different storage technologies over the life cycle of larger data sets. Since they reduce the operator effort for data movements, they can diminish the risk of data loss due to human manipulation errors. Typical examples of HSM systems for data life cycle management are archive and backup systems, which manage access to one or more consistent data snapshots across different storage technologies.

Additional end-user benefits are a single consistent system interface and a stable abstraction from a, potentially evolving, back-end implementation of the storage system. This abstraction is particularly important in large shared storage installations, which usually need to operate without downtime and hence require the replacement of faulty components and storage media to take place with minimal impact on normal operation.

Automated data migration between storage technologies can conceptually take place at different data granularity – from small groups of data blocks within a user file to large groups of files that span multiple tape volumes. The dominating implementation choice though are systems that consider complete, individual data files as management granularity.

Storage Hierarchies for Big Data, Fig. 1 Memory and storage hierarchy



Hybrid Disk Drives and Multitiered File Systems

Conceptually similar, but at small volume scale, also hybrid disk drives implement a tiered storage system. These devices combine the fast random access characteristic of NAND-flash with the more economical storage capacity of magnetic disks in a single unit. Based on the measured access patterns, the device dynamically replicates often used data blocks in its small flash cache area, which can greatly improve application performance without adverse effects on volume cost or total unit capacity. These hybrid devices are most useful in deployments with space or connectivity constraints (e.g., laptops and small or embedded servers).

In contexts free of the above constraints, a similar hybrid combination can be constructed via multitier file systems such as FusionDrive (Apple Core Storage) or Hybrid Storage Pools in ZFS (Bonwick and Moore, 2003; Gregg, 2009) that combine the fast access of NAND-flash with the large capacity of magnetic disks.

Challenges for Hierarchical Storage Systems

HSM systems can greatly improve resource utilization in particular in environments with stable, organized workflows and established prioritization between concurrent applications.

In more complex environments, with many concurrent interdependent applications and many users with differing or conflicting priorities, HSM system can show visible limitations. In these cases both storage system operation and user experience become complex and less predictable – up to the point that the HSM automation benefits are out-weighted by additional human effort to stabilize operation (by manually partitioning the resources and throttling concurrent user activity) and artificial storage operations by the user side to “trick” the system into the desired behavior.

A further challenge in large-scale deployments can arise from a mismatch between the granularity of HSM data movements (e.g., individual files) and the conceptual entities of a user workflow (e.g., larger groups of files in a typical big data application). This granularity mismatch

leads to difficulties let HSM data migrations appear as complete, atomic operations in the problem space of the user. A HSM with a purely file-based interface can only incomplete information of the user intention to access (or abandon) a larger group of files. The system may hence fail to properly predict the resource and time requirements to complete the intended operation or to predict the interference with data movements on behalf of other users/applications in a shared storage system.

Storage Hierarchies in Practical Applications

Tiered storage and hierarchical storage systems are today used in a variety of environments and are one of the key mechanisms to construct large-scale, cost-efficient storage systems that can adapt to changing access patterns over the lifetime of larger data set deployments. Data-intensive sciences such as high-energy physics, astrophysics, astronomy, and life sciences have constructed massive scientific data repositories (Bird et al., 2005; Pace, 2014) of several hundreds of petabytes by combining robotic tape archives, distributed disk clusters, and fast solid-state storage. The availability of reliable, high-bandwidth networks enabled to build large, distributed data repositories that enable collaborative scientific discovery and result sharing across dispersed communities. In data-intensive sciences, resource efficiency is of particular importance since computing budget constraints for storage media could limit the achievable statistical precision and hence relevance of a study.

On the commercial side during the last decade, large data amounts from legacy sources and databases and an increasing volume of environmental or commercial sensors have become the basis for many analytical applications. Hadoop with HDFS as location-aware storage environment for parallel processing and S3 as archive storage with disk or tape as back-end have become increasingly popular. Recently also, the large available amount of directly

addressable memory has allowed to move previously big data analytical problems entirely into memory or nonvolatile solid-state storage to avoid the latency of magnetic media access and TCP networking round trips for much of their execution. In this context Apache Spark (Zaharia et al., 2010) plays an important role in combining parallel data selection on a large disk-based repository with large distributed in-memory caches in the same run-time environment.

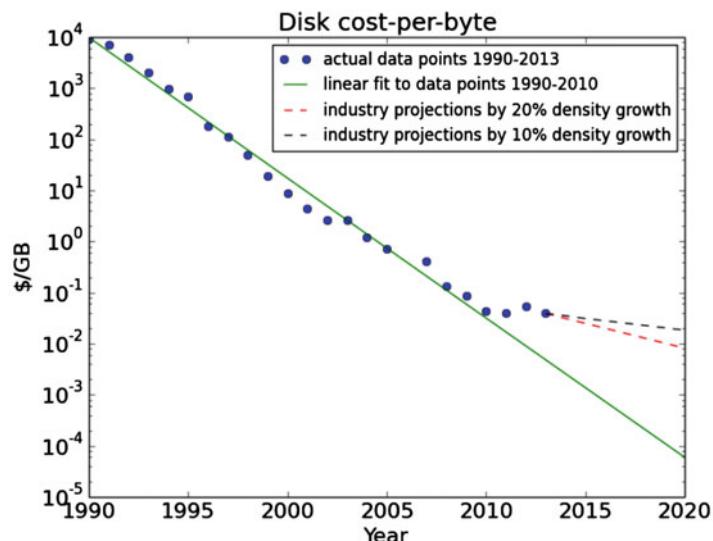
Expected Technology and Market Changes

The established role of storage hierarchies in big data applications is expected to continue also in the future, when several new technologies will become available and market changes will shift the price balance between the available media types. On the magnetic disk side, one can since several years observe a slowdown of the areal media density evolution (see Fig. 2 from Kryder rate Walter, 2005; Gupta et al., 2014; Mellor, 2014; Klein, 2017) as the R&D effort increases to productize new magnetic recording techniques like shingled magnetic recording (Feldman and Gibson, 2013) (SMR), microwave-assisted magnetic recording (Zhu et al., 2007) (MAMR), and

heat-assisted magnetic recording (Kryder et al., 2008) (HAMR).

Recording technologies such as SMR further imply a shifted balance between update and read performance, which will limit scalability in some use cases, unless combined with fast memory- or flash-based cache. Another influence on the media price balance may arise from the increasing consolidation of magnetic disk deployments to only a few, larger cloud providers, who are seeking increased influence on the design of magnetic disk units (Brewer et al., 2016). For the small number of vendors in the disk market, it will likely become more important to suit large computer center deployments than to maintain compatibility with the traditional form factor of previous desktop computer generations. The home and mobile markets are already largely based on devices with NAND-flash storage, which will see increasing competition from direct accessible persistent memory with higher durability and reduced write amplification (Intel 2015; <https://arstechnica.com/information-technology/2017/03/intels-first-optane-ssd-375gb-that-you-can-also-use-as-ram/>). This storage technology, if their advertised price point and functional capabilities are confirmed, would allow to further extend the problem space reachable by in-memory processing. Together with DRAM these new media would also enable

Storage Hierarchies for Big Data, Fig. 2 Magnetic disk price evolution



faster hybrid storage devices with NAND-flash or magnetic disks. Tape at the high volume end of the storage hierarchy has but due to very low media and power cost been remarkably successful – despite its for many applications problematic combination of high sequential rate and high access latency. Also here, visible market consolidation is influencing the continued technology evolution, and archive storage hierarchies involving tape will get increasing competition from disk- and flash-based alternatives (Gupta et al., 2014).

Conclusion

In conclusion, the use of tiered or hierarchical media combinations from several layers of the storage hierarchy will continue, and the recent fast, high-density, direct-access solid-state storage will allow to enlarge the problem size accessible to big in-memory applications. For problems that require larger volume scalability, the parallel cloud processing environments will benefit from some continued media density (and hence volume) growth but at lower Kryder rate. The performance gap between sequential read and other access methods (e.g., random access, write/update access) will continue to increase on the magnetic disk side. The benefits of integrating magnetic media with direct-access solid-state storage will therefore for many big data applications get even more important.

References

- Bird I et al (2005) LHC computing grid—technical design report. CERN-LHCC-2005-024
- Bonwick J, Moore B (2003) ZFS: the last word in file systems. http://opensolaris.org/os/community/zfs/docs/zfs_last.pdf
- Brewer E et al (2016) Disks for data centers. <https://research.google.com/pubs/pub44830.html>
- Feldman T, Gibson G (2013) Shingled magnetic recording—areal density increase requires new data management. *Login* 38(3):22–30
- Gregg B (2009) Hybrid storage pool: top speeds. <http://dtrace.org/blogs/brendan/2009/10/08/hybrid-storage-pool-top-speeds/>
- Gupta P et al (2014) An economic perspective of disk vs. flash media in archival storage. In: IEEE MASCOTS 2014
- Intel (2015) Micron debut 3D XPoint storage technology 1,000x faster than current SSDs. <https://www.cnet.com/news/intel-and-micron-debut-3d-xpoint-storage-technology-thats-1000-times-faster-than-existing-drives/>
- Klein A (2017) Hard disk cost per gigabyte. <https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/>
- Kryder et al (2008) Heat assisted magnetic recording. *Proc IEEE* 96(11):1810–1835
- Mellor C (2014) Kryder's law craps out: race to UBER-CHEAP STORAGE is OVER. https://www.theregister.co.uk/2014/11/10/kryders_law_of_ever_cheaper_storage_disproven/
- Pace A (2014) Technologies for large data management in scientific computing. *Int J Mod Phys C* 25(2):1430001
- Walter C (2005) Kryder's law. *Sci Am* 293(2):32–3
- Zaharia M et al (2010) Spark: cluster computing with working sets. Technical report No. UCB/EECS-2010-53, University of California, Berkeley
- Zhu J, Zhu X, Tang Y (2007) Microwave Assisted Magnetic Recording, IDEMA

Storage Technologies for Big Data

Zhaojie Niu and Bingsheng He

National University of Singapore, Singapore,
Singapore

Overview

This article introduces big data storage systems. It first describes the development of storage techniques for big data. Then, it compares the different storage models and their most suitable use cases. Finally, it talks about the impact of emerging hardware on the big data storage techniques and introduces some hot research topics.

Introduction and Background

In the big data era, data are increasingly gathered from heterogeneous devices like IoT devices, sensor networks, scientific experiments, websites, and many other applications producing data in various formats. The data sets are

so voluminous and complex that traditional relational databases are inadequate to deal with them. Thus, innovative research and development of storage systems that can provide highly scalable, reliable, and efficient storage for dynamically increasing data is required. A movement to NoSQL (Not only SQL) occurs, and many NoSQL databases are proposed. They have simplicity of design, simpler “horizontal” scaling to clusters consists of thousands of servers, and more fine-grained control over availability. The data structures used by NoSQL databases are optimized for target data formats and sources, which are different from those used in relational databases, making some operations faster in NoSQL. Many NoSQL stores compromise consistency (in the context of CAP theory (Seth and Nancy, 2002)) in favor of partition tolerance, availability, and performance. These NoSQL stores also lack true ACID transaction supported by traditional relational databases, although a few databases, such as Google spanner (David et al., 2017), start to embrace the relational model.

Relational databases have been widely used efficiently for transaction processing in terms of storage and processing for many decades. The relational databases support ACID transaction that also limits their horizontal scalability. To achieve availability and better scalability, most NoSQL databases compromise consistency and do not support true ACID transaction support as traditional relational databases. The strengths and weakness for traditional relational database and NoSQL database are summarized in Table 1.

The NoSQL databases mainly consists four types of data models: key-value, column-oriented, document-oriented, and graph. These NOSQL systems are being used to handle exponentially growing data but cannot tolerate the performance degradation caused by data persistence. On the other hand, these systems face the challenges posed by the inherent difficulties in scaling DRAM and the cost of the DRAM. To address these challenges, new emerging hardware technologies like nonvolatile main memory (NVRAM) have lately received a great deal of attention in the database community. NVRAM is positioned between DRAM and secondary storage (such as SSD), both in terms of performance and cost. NVRAM can fundamentally change in-memory databases as data structures do not have to be explicitly backed up to hard drives or SSDs but can be inherently persistent in main memory (David et al., 2015; Jasmina et al., 2015; Mihnea et al., 2017; Sudarsun et al., 2016; Colaso et al., 2017). In section “[Taxonomy of Big Data Storage Technologies](#)”, we introduce these NoSQL systems based on the data model and how the emerging hardware impacts the design of the NOSQL systems.

Taxonomy of Big Data Storage Technologies

NoSQL databases mainly consist four types of data models: key-value, column-oriented,

Storage Technologies for Big Data, Table 1 Strengths and weakness for traditional relational database and NoSQL database

DB	Strength	Weakness
Traditional relational database	Support highly structured data; Efficient storage and processing in auxiliary server; ACID transaction support; Vertical scalability; Specialized data manipulation languages; Specialized data schema	Performance and processing delay bottleneck with growth of data; ACID support which hinders scalability; Limitation for schema-less and unstructured data
NoSQL database	Support heterogeneous structured data; Horizontal scalability with extendible commodity servers; High reliability; High availability	No compliance with ACID in order to achieve high scalability and high availability for most NoSQL databases

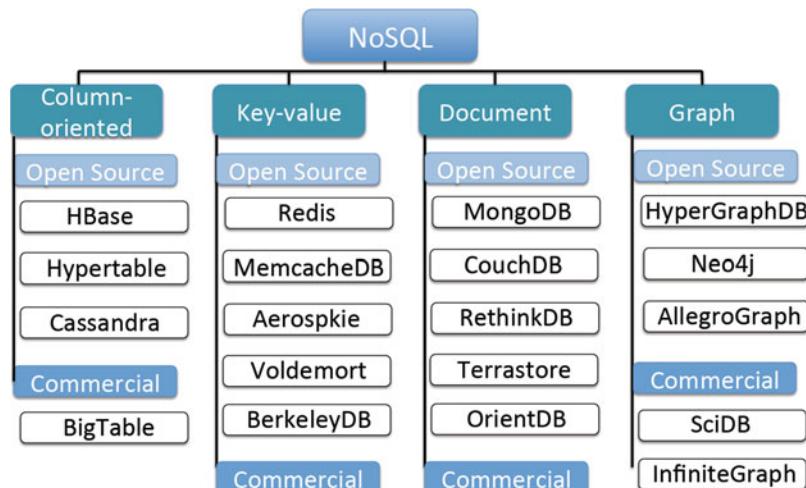
Storage Technologies for Big Data, Table 2

Performance, scalability, flexibility and complexity of different data models for NoSQL databases

Data model	Performance	Scalability	Flexibility	Complexity
Key-value	High	High	High	None
Column-oriented	High	High	Moderate	Low
Document-oriented	High	Variable	High	Low
Graph	Variable	Variable	High	High
Relational	Variable	Variable	Low	Moderate

Storage Technologies for Big Data, Fig. 1

Taxonomy of big data storage technologies based on data models



document-oriented, and graph (Michael, 2014). The performance, scalability, flexibility, and complexity for different data models are different that is shown in Table 2. We classify NoSQL databases based on the data models and introduce some representing systems for each category.

We classify the current representing big data storage system according to data models as shown in Fig. 1 and introduce them one by one in detail. We also discuss the recent development on emerging storage techniques with nonvolatile memory (NVRAM).

Key-Value Data Model

The key-value type, basically, uses a hash table in which there exist a unique key and a pointer to a particular item of data. A bucket is a logical group of keys, but they don't physically group the data. There can be identical keys in different buckets. This in-memory data structure design needs to explicitly backup in case of system crashes in DRAM which degrades the

performance. NVRAM guarantees data consistency even across power outages while it comes with new challenges as the consistency and persistence of in-memory data structures have to be guaranteed even across system crashes. Data structure and algorithms of the hash table especially on NVRAM need to be redesigned for the use in NV-persisted in-memory databases (David et al., 2015).

S

Column-Oriented Data Model

In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows. To increase the scalability and reduce the cost of using DRAM, SAP starts to explore the new column-based in-memory data structures and efficient algorithms

which are directly stored and used in NVRAM blocks (Mihnea et al., 2017).

Document-Oriented Data Model

The data which is a collection of key-value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as “documents”) provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), and BSON (which is a binary encoding of JSON objects) are some common standard encodings. In order to provide efficient memory access for these objects, cost-effective memory placement mechanisms are proposed on NVRAM (Sudarsun et al., 2016).

Graph Data Model

In a Graph Base NoSQL database, you will not find the rigid format of SQL or the tables and columns representation; a flexible graphical representation is instead used which is perfect to address scalability concerns. Graph structures are used with edges, nodes, and properties which provide index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database. With the increase of the scale of the graph, the capacity of scaling of DRAM is not able to satisfy the requirements of analytics over graph with trillions of connections. Explores of replacing DRAM with emerging nonvolatile memories (NVRAM) on large-scale graph analytics framework are performed. Some hybrid memory systems with NVRAM and DRAM are proposed in a cost-effective manner and the performance it can achieve is close to DRAM-only performance (Jasmina et al., 2015).

References

Andrei M, Lemke C, Radestock G, Schulze R, Thiel C, Blanco R, Meghlan A, Sharique M, Seifert S, Vishnoi S, Booss D, Peh T, Schreter I, Thesing W, Wagle M, Willhalm T (2017) Sap HANA adoption of non-volatile memory. Proc VLDB Endow 10(12):1754–1765

- Bacon DF, Bales N, Bruno N, Cooper BF, Dickinson A, Fikes A, Fraser C, Gubarev A, Joshi M, Kogan E, Lloyd A, Melnik S, Rao R, Shue D, Taylor C, van der Holst M, Woodford D (2017) Spanner: becoming a SQL system. In: Proceedings of the 2017 ACM international conference on management of data (SIGMOD’17). ACM, pp 331–343
- Colaso A, Prieto P, Abad P, Herrero JA, Menezo LG, Puente V, Gregorio JA (2017) Memory hierarchy characterization of NoSQL applications through full-system simulation. IEEE Trans Parallel Distrib Syst PP(99):1–1
- Gilbert S, Lynch N (2002) Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59
- Kannan S, Gavrilovska A, Schwan K (2016) PVM: persistent virtual memory for efficient capacity scaling and object storage. In: Proceedings of the eleventh European conference on computer systems (EuroSys’16). ACM, pp 13:1–13:16
- Malicevic J, Dulloor S, Sundaram N, Satish N, Jackson J, Zwaenepoel W (2015) Exploiting NVM in large-scale graph analytics. In: Proceedings of the 3rd workshop on interactions of NVM/FLASH with operating systems and workloads (INFLOW’15). ACM, pp 2:1–2:9
- Mior MJ (2014) Automated schema design for nosql databases. In: Proceedings of the 2014 SIGMOD PhD symposium (SIGMOD’14). ACM, pp 41–45
- Schwalb D, Dreseler M, Uflacker M, Plattner H (2015) Nvc-hashmap: a persistent and concurrent hashmap for non-volatile memories. In: Proceedings of the 3rd VLDB workshop on in-memory data management and analytics (IMDM’15). ACM, pp 4:1–4:8

Straight-Line Context-Free Tree Grammars

► Grammar-Based Compression

Straight-Line Programs

► Grammar-Based Compression

Stratosphere Platform

► Apache Flink

Stream Benchmarks

Jeyhun Karimov
DFKI GmbH, Berlin, Germany

Synonyms

Stream performance evaluation

Definitions

Stream benchmarks deal with performance evaluation techniques and define related metrics for stream data processing systems.

Overview

Firstly, we discuss background information on database benchmarking, foundations, main metrics, and main features for stream data benchmarking. Then, we provide related stream benchmarks and categorize them with respect to the application area.

Historical Background

Lee et al. (1997) initiated one of the first works in the related area, MediaBench, by evaluating and synthesizing multimedia and communications systems. Abadi et al. (2003) and Motwani et al. (2003) pioneered one of the first stream data processing systems, Aurora and STREAM, respectively. The need to compare the performance characteristics of streaming systems relative to each other and to alternative (e.g., Relational Database) systems endorsed the development of Linear Road benchmark (Arasu et al., 2004).

Foundations

Stream data processing is key when the large volumes of input data have to be processed fast

to quickly adapt and react to changes. Therefore, stream data processing has gained significant attention.

The main intuition behind stream benchmarks is to define a standard to compare streaming systems, which has different characteristics, in a various use-cases. Stream benchmarks simulate an environment with different workloads and analyze the behavior of the systems to be tested. The more similar the benchmark to the real production environment, the more realistic and valuable it is.

In the high level, stream benchmark framework consists of two main components: system under test and driver. Driver is responsible for simulating the production environment w.r.t. a given use-case. System under test is actual tested streaming system which can be out-of-the-box or tuned.

Accurately representing the system under test is important when designing benchmarks. Schroeder et al. (2006) categorize benchmarks into closed, open, and partly open models. In a closed system model, the input arrivals are triggered after the completion of the previous input processing and some thinking time delay. In an open system model, on the other hand, new input arrivals and the completion of the previous input processing are independent. In a partly open system model, we specify the settings for which partly open model behaves like a closed or an open model.

Below, we categorize existing stream benchmarks in literature. Firstly, we talk about the main metrics in stream benchmarks and their definitions. Secondly, we analyze stream benchmarks which concentrate on specific features such as fault tolerance and state management. Lastly, we analyze stream benchmarks built for specific industrial use-cases.

Metrics

Main metrics for stream benchmarks are latency and throughput. Achieving high throughput while preserving low latency is the main goal for streaming systems.

Latency

Defining a standard latency metric definition for streaming systems is a challenging task. One reason is significant architectural and computational differences among stream data processing engines. Another reason is calculating latency for stateful operators in nontrivial.

Chintapalli et al. (2016a) use Redis for stateful computations as part of the system under test. The authors calculate the event-time latency by subtracting the window start time and duration time from the last updated time of a particular input record. Perera et al. (2016) propose reproducible benchmarks for Apache Spark and Flink on public clouds. The authors do not use a standard latency definition, as, in this case, the latency measurement is experiment and application specific. Lu et al. (2014) propose a new stream benchmark by separating the data generator and system under test. The authors put a mediator layer between the two components. They define latency as an average time span from the arrival of a record till the end of processing of the record. Qian et al. (2016) also adopted a similar approach. Karimov et al. (2018) develop stream benchmark framework that overcome overhead of a mediator layer. The authors show how processing-time latency might mislead when compared with event-time latency.

Throughput

Throughput is another essential metric for stream data processing systems. Similar to latency, measuring and defining a standard throughput metric for all streaming systems is nontrivial.

Chintapalli et al. (2016a) separate the data generator and system under test with an intermediate layer between them. The authors calculate the throughput by configuring the speed of data generator for a specific workload. Lopez et al. (2016a), on the other hand, rely on Kafka's sampled throughput rates. Lu et al. (2014) measure the overall system under test throughput and throughput per node. The authors define overall throughput metric by count (average count of records per second) and size (average data size in terms of bytes processed per second). Shukla and Simmhan (2016) define throughput as the rate of

output tuples emitted from the output operators in a unit time. Dayarathna and Suzumura (2013) define job throughput in two ways. First, the authors measure the time required to process a specific amount of events. Second, the authors measure the number of tuples processed in a given amount of time. The throughput computation is performed based on both time periods. Samosir et al. (2016), on the other hand, adopt the throughput metric used in batch processing systems. Karimov et al. (2018) propose maximum sustainable throughput throughout the whole experiment.

Benchmarking the energy consumption of stream data processing engines is another important aspect of stream benchmarks. Dayarathna et al. (2017) adopt Linear Road benchmark for testing the energy efficiency of S4, Storm, ActiveMQ, Esper, Kafka, and Spark Streaming. The key finding of this work is that better power consumption behaviors in the context of data stream processing systems can be achieved by setting tuple sizes to be moderate and scheduling plans to have balanced system overhead.

Features

Besides focusing on metrics computations, stream benchmarks also concentrate on specific features of stream data processing engines.

Fault Tolerance

Lopez et al. (2016a) study streaming systems' tolerance to failures by analyzing the system behavior after detecting the failure. The system behavior includes the message losses and latency/throughput change during node failure. Gradvohl et al. (2014) categorize fault tolerance behavior in stream data processing systems into replication components, upstream backup, checkpoint, and recovery and analyze the system utilization of these strategies. Mohamed et al. (2017) propose a driver which allows programmatic specification of complex fault scenarios. Chauhan et al. (2012) measure the systems' tolerance to faults by (i) measuring the

number of events handled and (ii) checking the number of events that were missed when nodes go down in cluster. Qian et al. (2016) adopt identity workload and consider only one node failure at a time. The benchmark suite collects the performance metrics in node-failure workload and compares it with non-faulty workload.

State Management

Linear Road benchmark, Arasu et al. (2004) and Jain et al. (2006), consists of continuous queries which update operator state by processing the incoming stream. Kipf et al. (2017) analyze the limitation of efficiently exposing the state to analytical queries for stream data processing systems. The authors compare main-memory databases and streaming engines and propose new methods to advance state management in streaming systems.

Key Applications

In this section we categorize stream benchmarks based on different industrial use-cases.

Data mining. Zhang et al. (2012) and Le-Phuoc et al. (2012) are the pioneers to propose SRBench and LSBench, the first benchmarks for RDF streaming and Linked Stream Data processing. The authors adopt wide range of queries including simple graph pattern matching queries and the ones with complex reasoning tasks. DellAglio et al. (2013) propose CSRBM, an extension for SRBench. The authors overcome the main shortcoming of SRBench and LSBench, which is inability to assess the correctness of query evaluation results, by analyzing the operational semantics of the particular processors. Ali et al. (2015) address another limitation of SRBench and LSBench, addressing the dynamic application requirements and data-dependent properties. The authors propose the workloads which include fluctuating streaming rates during query execution and changing the application requirements over a some time duration. Implementing, modeling, and evaluating the provisioning algorithms for stream processing applications is

another related work, in which authors propose VISIP Testbed (Hochreiner, 2017). The toolkit provides a common runtime for stream processing applications.

E-commerce. Teng et al. (2017) analyze streaming system behavior in e-commerce scenarios. The authors provide a data generator, as part of the benchmark suite, with certain user models, which adopt a certain user habits in e-commerce platforms. Tucker et al. (2008) propose NEXMark, an extension of XMark, Schmidt et al. (2001), based on online auction system. Currently, NEXMark is used as a benchmark suite in Apache Beam, Buzzwords (2017).

IoT. Shukla and Simmhan (2016) develop a benchmark suite for streaming systems for IoT applications. The authors classify 13 common IoT tasks with functional categories. Moreover, the benchmark suite provides two IoT applications being statistical summarization and predictive analytics. CityBench is the benchmark to evaluate RDF stream processing systems in IoT scenarios, Ali et al. (2015). The authors use traffic vehicles, parking, weather, pollution, cultural, and library events, with changing event rates and playback speeds as part of the data generator. Shukla et al. (2017) extend the existing stream benchmarks in IoT proposing RIoTBench. The benchmark includes 27 common IoT tasks. Moreover, the authors propose four IoT application benchmarks composed from the proposed tasks.

Network. Nazhandali et al. (2005) propose stream benchmark for sensor network systems, suitable for sensor processors. The authors propose new metrics being EPB (Energy Per Bundle) and CFP (Composition Footprint) to evaluate and compare systems under test. Lopez et al. (2016b) analyze the performance of Virtualized Network Function for real-time thread detection using stream processing. Wolf and Franklin (2000) propose telecommunication benchmark for network processors. The authors adopt four workloads for data stream processing in telecommunications scenario. Trivedi et al.

(2016) analyze yet another interesting aspect, the (ir)relevance of network bandwidth to modern streaming engines. The key finding of this paper is that current streaming engines need significant architectural improvements as they cannot benefit from high bandwidth networks.

Multi-core processors. Zhang et al. (2017) benchmark the current design of stream data processing engines on multi-core processors analyzing the possible bottlenecks of massively parallel JVM-based streaming engines.

CEP. Mendes et al. (2009) were among the pioneers to propose a benchmark for CEP systems. The authors provide series of queries to exercise factors such as window size and policy, selectivity, and event dimensionality. Mendes et al. (2013) propose BiCEP, the domain-specific benchmark suite, to evaluate different performance aspects of event processing platforms. Alevizos and Artikis (2014) adopt existing techniques to analyze widely used Esper system which employs a SQL-based language and RTEC which is a dialect of the Event Calculus.

Machine learning. Gama et al. (2009) propose a general framework for assessing predictive stream learning algorithms. Gama et al. (2013) focus on decision models and develop a benchmark suite to evaluate continuously evolving streaming and to detect and react to real-time input data. Imai et al. (2017) utilize a machine learning model to predict the maximum sustainable throughput in streaming systems.

Bottlenecks. When designing a system, it is important to detect and avoid bottlenecks. For streaming systems, Chintapalli et al. (2016b) show the Zookeeper being a main performance bottleneck for Storm. For stream benchmarks, Friedrich et al. (2017) show the limitations of existing benchmark designs and the possible biased results. Moreover, Artisans (2017) makes a disclaimer for the original implementation of Chintapalli et al. (2016a), showing an intermediate message layer, Kafka,

and external state management system, Redis, are actually being a bottleneck for Flink's overall performance.

Cross-References

► Distributed Systems

References

- Abadi DJ, Carney D, Çetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. VLDB J Int J Very Large Data Bases 12(2):120–139
- Alevizos E, Artikis A (2014) Being logical or going with the flow? A comparison of complex event processing systems. In: SETN. Springer, pp 460–474
- Ali MI, Gao F, Mileo A (2015) Citybench: a configurable benchmark to evaluate RSP engines using smart city datasets. In: International semantic web conference. Springer, pp 374–389
- Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Proceedings of the thirtieth international conference on very large data bases, vol 30. VLDB Endowment, pp 480–491
- Artisans D (2017) Extending the Yahoo! streaming benchmark. <https://data-artisans.com/blog/extending-the-yahoo-streaming-benchmark>. Online: Accessed 1 Nov 2017
- Buzzwords B (2017) Nexmark: using apache beam to create a unified benchmarking suite. https://berlin-buzzwords.de/sites/berlinbuzzwords.de/files/media/documents/nexmark_suite_for_apache_beam_berlin_buzzwords_1.pdf. Online: Accessed 1 Nov 2017
- Chauhan J, Chowdhury SA, Makaroff D (2012) Performance evaluation of yahoo! s4: a first look. In: 2012 seventh international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC). IEEE, pp 58–65
- Chintapalli S, Dagit D, Evans B, Farivar R, Graves T, Holderbaugh M, Liu Z, Nusbaum K, Patil K, Peng BJ et al (2016a) Benchmarking streaming computation engines: storm, flink and spark streaming. In: 2016 IEEE international parallel and distributed processing symposium workshops. IEEE, pp 1789–1792
- Chintapalli S, Dagit D, Evans R, Farivar R, Liu Z, Nusbaum K, Patil K, Peng B (2016b) Pacemaker: when zookeeper arteries get clogged in storm clusters. In: 2016 IEEE 9th international conference on cloud computing (CLOUD). IEEE, pp 448–455

- Dayarathna M, Suzumura T (2013) A performance analysis of system s, s4, and esper via two level benchmarking. In: International conference on quantitative evaluation of systems. Springer, pp 225–240
- Dayarathna M, Li Y, Wen Y, Fan R (2017) Energy consumption analysis of data stream processing: a benchmarking approach. *Softw Pract Exp* 47(10): 1443–1462
- DellAglio D, Calbimonte JP, Balduini M, Corcho O, Della Valle E (2013) On correctness in RDF stream processor benchmarking. In: International semantic web conference. Springer, pp 326–342
- Friedrich S, Wingerath W, Ritter N (2017) Coordinated omission in NoSQL database benchmarking. In: BTW (Workshops), pp 215–225
- Gama J, Sebastião R, Rodrigues PP (2009) Issues in evaluation of stream learning algorithms. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 329–338
- Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. *Mach Learn* 90(3): 317–346
- Gradvohl ALS, Senger H, Arantes L, Sens P (2014) Comparing distributed online stream processing systems considering fault tolerance issues. *J Emerg Technol Web Intell* 6(2):174–179
- Hochreiner C (2017) Visp testbed-a toolkit for modeling and evaluating resource provisioning algorithms for stream processing applications. *Strategies* 1(6):9–17
- Imai S, Patterson S, Varela CA (2017) Maximum sustainable throughput prediction for data stream processing over public clouds. In: Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing. IEEE Press, pp 504–513
- Jain N, Amini L, Andrade H, King R, Park Y, Selo P, Venkatramani C (2006) Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. ACM, pp 431–442
- Karimov J, Rabl T, Katsifodimos A, Samarev R, Heiskanen H, Markl V (2018) Benchmarking distributed stream data processing systems. In: ICDE
- Kipf A, Pandey V, Böttcher J, Braun L, Neumann T, Kemper A (2017) Analytics on fast data: main-memory database systems versus modern streaming systems. In: EDBT, pp 49–60
- Le-Phuoc D, Dao-Tran M, Pham MD, Boncz P, Eiter T, Fink M (2012) Linked stream data processing engines: facts and figures. In: The semantic Web-ISWC 2012, pp 300–312
- Lee C, Potkonjak M, Mangione-Smith WH (1997) Media-bench: a tool for evaluating and synthesizing multimedia and communications systems. In: Proceedings of the 30th annual ACM/IEEE international symposium on microarchitecture. IEEE Computer Society, pp 330–335
- Lopez MA, Lobato AGP, Duarte OCM (2016a) A performance comparison of open-source stream processing platforms. In: 2016 IEEE global communications conference (GLOBECOM). IEEE, pp 1–6
- Lopez MA, Lobato AGP, Duarte OCM, Pujolle G (2016b) Design and performance evaluation of a virtualized network function for real-time threat detection using stream processing
- Lu R, Wu G, Xie B, Hu J (2014) Stream bench: towards benchmarking modern distributed stream computing frameworks. In: 2014 IEEE/ACM 7th international conference on utility and cloud computing (UCC). IEEE, pp 69–78
- Mendes MR, Bizarro P, Marques P (2009) A performance study of event processing systems. In: Technology conference on performance evaluation and benchmarking. Springer, pp 221–236
- Mendes M, Bizarro P, Marques P (2013) Towards a standard event processing benchmark. In: Proceedings of the 4th ACM/SPEC international conference on performance engineering. ACM, pp 307–310
- Mohamed S, Forshaw M, Thomas N, Dinn A (2017) Performance and dependability evaluation of distributed event-based systems: a dynamic code-injection approach. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering. ACM, pp 349–352
- Motwani R, Widom J, Arasu A, Babcock B, Babu S, Datar M, Manku G, Olston C, Rosenstein J, Varma R (2003) Query processing, resource management, and approximation in a data stream management system. In: CIDR
- Nazhandali L, Minuth M, Austin T (2005) Sensebench: toward an accurate evaluation of sensor network processors. In: Proceedings of the IEEE international workload characterization symposium, 2005. IEEE, pp 197–203
- Perera S, Perera A, Hakimzadeh K (2016) Reproducible experiments for comparing apache flink and apache spark on public clouds. arXiv preprint arXiv:161004493
- Qian S, Wu G, Huang J, Das T (2016) Benchmarking modern distributed streaming platforms. In: 2016 IEEE international conference on industrial technology (ICIT). IEEE, pp 592–598
- Samosir J, Indrawan-Santiago M, Haghghi PD (2016) An evaluation of data stream processing systems for data driven applications. *Proc Comput Sci* 80:439–449
- Schmidt AR, Waas F, Kersten ML, Florescu D, Manolescu I, Carey MJ, Busse R (2001) The XML benchmark project
- Schroeder B, Wierman A, Harchol-Balter M (2006) Open versus closed: a cautionary tale. In: NSDI, vol 6, pp 18–18
- Shukla A, Simmhan Y (2016) Benchmarking distributed stream processing platforms for IoT applications. In: Technology conference on performance evaluation and benchmarking. Springer, pp 90–106
- Shukla A, Chaturvedi S, Simmhan Y (2017) Riotbench: a real-time IoT benchmark for distributed stream processing platforms. arXiv preprint arXiv:170108530

- Teng M, Sun Q, Deng B, Sun L, Qin X (2017) A tool of benchmarking realtime analysis for massive behavior data. In: Asia-Pacific web (APWeb) and web-age information management (WAIM) joint conference on web and big data. Springer, pp 345–348
- Trivedi A, Stuedi P, Pfefferle J, Stoica R, Metzler B, Kotsidas I, Ioannou N (2016) On the [ir] relevance of network performance for data processing. Network 40:60
- Tucker P, Tufte K, Papadimos V, Maier D (2008) Nemmark—a benchmark for queries over data streams (draft). Technical report, OGI School of Science & Engineering at OHSU
- Wolf T, Franklin M (2000) Commbench—a telecommunications benchmark for network processors. In: 2000 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, pp 154–162
- Zhang Y, Duc PM, Corcho O, Calbimonte JP (2012) Srbench: a streaming RDF/SPARQL benchmark. In: International semantic web conference. Springer, pp 641–657
- Zhang S, He B, Dahlmeier D, Zhou AC, Heinze T (2017) Revisiting the design of data stream processing systems on multi-core processors. In: 2017 IEEE 33rd international conference on data engineering (ICDE). IEEE, pp 659–670

Stream Learning

- ▶ [Online Machine Learning Algorithms over Data Streams](#)
- ▶ [Online Machine Learning in Big Data Streams: Overview](#)
- ▶ [Recommender Systems over Data Streams](#)
- ▶ [Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning](#)

Stream Performance Evaluation

- ▶ [Stream Benchmarks](#)

Stream Processing Language

- ▶ [Stream Processing Languages and Abstractions](#)

Stream Processing Languages and Abstractions

Martin Hirzel and Guillaume Baudart
IBM Research AI, Yorktown Heights, NY, USA

Synonyms

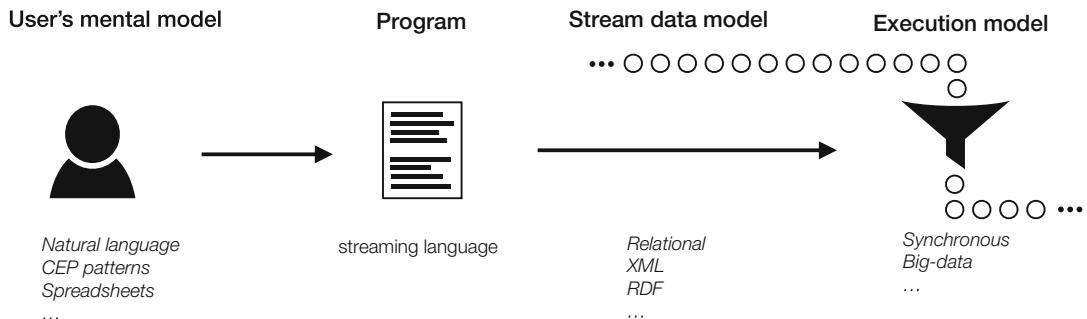
[Continuous dataflow language](#); [Streaming language](#); [Stream processing language](#)

Definitions

A *stream processing language* is a programming language for specifying streaming applications. Here, a *stream* is an unbounded sequence of data items, and a *streaming application* is a computer program that continuously consumes input streams and produces output streams. This article surveys recent streaming languages designed around the user's mental model, the stream data model, or the execution model, as illustrated in Fig. 1. In addition to specific languages, this article also discusses *abstractions* for stream processing, which are high-level language constructs that make it easy to express common stream processing tasks.

Overview

Continuous data streams arise from many directions, including sensors, communications, and commerce. Stream processing helps when low-latency responses are of the essence or when streams are too big to store for offline analysis. Programmers can of course write streaming applications in a general-purpose language without resorting to a dedicated domain-specific language (DSL) for streaming. However, using a streaming language makes code easier to read, write, understand, reason about, modularize, and optimize. Indeed, a suitable streaming language can help developers conceive of a solution to their streaming problems.



Stream Processing Languages and Abstractions, Fig. 1 Stream processing languages

This article provides definitions, surveys concepts, and offers pointers for more in-depth study of recent streaming languages. The interested reader may also want to refer to earlier papers for historic perspective: the 1997 survey by Stephens focuses on streaming languages (Stephens, 1997), the 2002 survey by Babcock et al. focuses on approximate streaming algorithms (Babcock et al., 2002), and the 2004 survey by Johnston et al. addresses dataflow languages, where streaming is a special case of dataflow (Johnston et al., 2004).

The central abstractions of stream processing are streams, operators, and stream graphs. A *stream* is an unbounded sequence of data items, for example, position readings from a delivery truck. A streaming *operator* is a stream transformer that transforms input streams to output streams. From the perspective of a streaming application, an operator can also have zero input streams (*source*) or zero output streams (*sink*). Finally, a *stream graph* is a directed graph whose nodes are operators and whose edges are streams. Some literature assumes that the shape of stream graphs is restricted, e.g., acyclic, but this article makes no such assumption. While only some streaming languages make the stream graph explicit, others use it as an intermediate representation. For example, the query plan generated from streaming SQL dialects is a stream graph.

The field of streaming languages is diverse and fast-moving. To understand where that diversity comes from, it is instructional to classify streaming languages by their *raison d'être*. Some streaming languages are based on the attitude

that since streams are data in motion, data is most central, and the language should be built around a data model (relational, XML, RDF). Other streaming languages focus on the execution model for processing the dataflows efficiently, by enforcing timing constraints or exploiting distributed hardware (synchronous, big-data). A third class of streaming languages focus more on enabling the end user to develop streaming applications in high-level or familiar abstractions (complex events, spreadsheets, or even natural language). Section “[Findings](#)” surveys languages in each of these classes, and section “[Examples](#)” gives concrete examples for two languages.

Findings

This section gives an overview of the field of stream processing languages by surveying eight prominent approaches. Each approach is exemplified by one concrete language. The approaches are grouped along the lines of the previous section into approaches driven by the data model, by the execution model, or by the target user.

Data-Model Driven Streaming Languages

The success of the **relational** data model for database systems has inspired streaming dialects of the SQL database language. These dialects benefit from developers' familiarity with SQL and from its relational algebra underpinnings. A prominent example is the CQL language, which complements standard relational operators with operators to transform streams into relations and

vice versa (Arasu et al., 2006). CQL lends itself to strong static typing, cf. Figure 10 of Soulé et al. (2016). Efforts toward standardizing streaming SQL focused on clarifying semantic corner cases (Jain et al., 2008).

The success of **XML** as a universal exchange format for events and messages has inspired XML-based streaming languages. These languages take advantage of a rich ecosystem of XML tools and standards and of the fact that XML documents are self-describing. The languages come in different flavors, from view maintenance over XML updates in NiagaraCQ (Chen et al., 2000) to languages that process streams where each data item is a (part of an) XML document (Diao et al., 2002; Mendell et al., 2012).

The Resource Description Framework (**RDF**) is a versatile data format for integration and reasoning, based on triples of the form `(subject, predicate, object)`. A popular language for querying static RDF knowledge bases is SPARQL (Prud'hommeaux and Seaborne, 2008), and C-SPARQL (Barbieri et al., 2009) extends SPARQL for continuous queries, just like CQL extends SQL. A stream is a sequence of timestamped triples, but a query can also return a graph by emitting multiple triples with the same timestamp.

Execution-Model Driven Streaming Languages

Dataflow **synchronous** languages (Benveniste et al., 2003) were introduced in the late 1980s as domain-specific languages for the design of embedded control systems. A dataflow synchronous program executes in a succession of discrete steps, and each step is assumed to be instantaneous (the synchronous hypothesis). A programmer writes high-level specifications in the form of stream functions specifying variable values at each step or instant. Section “[Synchronous Dataflow in Lustre](#)” illustrates this approach with the language Lustre (Caspi et al., 1987).

Streaming **big data** is motivated by the “4 Vs”: a lot of data (volume) streams quickly (velocity) into the system, which must deal with diverse data and functionality (variety) and with uncertainty (veracity). Languages for big-data

streaming let users specify an explicit stream graph that can be easily distributed with minimal synchronization and are extensible by operators in widely adopted general-purpose languages. Section “[Big-Data Streaming in SPL](#)” elaborates on this for the concrete example of SPL (Hirzel et al., 2017).

Target-User Driven Streaming Languages

Complex event processing, or **CEP**, lets users compose events hierarchically to span the gap between low-level and high-level concepts. There are various pattern languages for CEP that compile to finite automata. Recognizing this, the MATCH-RECOGNIZE SQL extension proposal simply adopts familiar regular expressions as the CEP pattern language (Zemke et al., 2007). While the SQL basis focuses on a relational model, regular expressions can also be used for CEP in big-data streaming (Hirzel, 2012).

Since there are many more **spreadsheet** users than software developers, a spreadsheet-based streaming language could reach more target users. Furthermore, spreadsheets are reactive: changes trigger updates to dependent formulas. ActiveSheets hooks up some spreadsheet cells to input or output streams, with normal spreadsheet formulas in between (Vaziri et al., 2014). When the two-dimensional spreadsheet data model is too limiting, it can be augmented with windows and partitioning (Hirzel et al., 2016).

A streaming language based on **natural language** might reach the maximum number of target users. However, since natural language is ambiguous, a controlled natural language (CNL) is a better choice (Kuhn, 2014). For instance, the language for META is a CNL for specifying event-condition-action rules, temporal predicates, and data types (Arnold et al., 2016). The data model includes events and entities with nested concepts and can be shown to be equivalent to the nested relational model (Shinnar et al., 2015).

Examples

This section gives details and concrete code examples for two out of the eight approaches for languages surveyed in the previous section:

the synchronous dataflow approach exemplified by Lustre (Caspi et al., 1987) and the big-data streaming approach exemplified by SPL (Hirzel et al., 2017).

When it comes to implementing streaming languages, there is a spectrum from basic to sophisticated techniques. At the basic end of the spectrum are configuration files in some existing markup format such as XML. The streaming engine interprets the configuration file to construct and then execute a stream graph. An intermediate point is a domain-specific embedded language (EDSL or sometimes DSEL) (Hudak, 1998). As the name implies, an EDSL is a domain-specific language (DSL) that is embedded in some host language, typically a general-purpose language (GPL). The line between simple libraries and EDSLs is blurred, but in general, EDSLs encourage a more idiomatic programming style. Recently, EDSLs have gained popularity as several GPLs have added features that make them more suitable for hosting EDSLs. At the sophisticated end of the spectrum are full-fledged, stand-alone DSLs with their own syntax, compiler, and other tools. While stand-alone streaming DSLs are not embedded in a GPL, they often interface with a GPL, e.g., for user-defined operators.

For clarity of exposition, the following examples use stand-alone streaming languages. Stand-alone languages are the norm for synchronous dataflow, because self-contained code is easier to reason about. On the other hand, for big-data streaming, EDSLs that specify an explicit stream graph are popular, because they are easier to implement. But as the example below illustrates, once implemented, stand-alone languages also have advantages for big-data streaming.

Synchronous Dataflow in Lustre

Synchronous dataflow languages were introduced to ease the design and certification of embedded systems by providing a well-defined mathematical framework that combines a logical notion of time and deterministic concurrency. It is then possible to formally reason about the system, simulate it, prove safety properties, and generate embedded code. The synchronous dataflow language Lustre is the backbone of the industrial lan-

guage and compiler Scade (Colaco et al., 2017) routinely used to program embedded controllers in many critical applications.

In Lustre a program is a set of equations defining streams of values. Time proceeds by discrete logical steps, and at each step, the program computes the value of each stream depending on its inputs and possibly previously computed values. Consider the example of Fig. 2 adapted from Bourke et al. (2017). The function *counter* takes three input streams, two integer streams *init* and *incr*, and one boolean stream *reset*. It returns the cumulative sum of the values of *incr* initialized with *init* and similarly reset when *reset* is *true* (Line 5). The variable *pn* (Line 4) stores the value of the counter *n* at the previous step using the initialization operator (\rightarrow) and the non-initialized delay **pre**.

A stream is not necessarily defined at each step. The clock of a stream is a boolean sequence giving the instants where it is defined. Streams with different clocks can be combined via sampling (**when**) or stuttering (**current**). For instance, the *tracker* function of Fig. 2 tracks the number of times the speed of a vehicle exceeds the speed limit. The **when** operator samples a stream according to a boolean condition. The function *counter* is thus only activated when *x* is *true* (Line 12). The **current** operator completes a stream with the last defined value when it is not present (Line 13). The value of *t* is thus sustained when *x* is *false*. The execution of such a program can be represented as a timeline, called a chronogram (illustrated in Fig. 2), showing the sequence of values taken by its streams at each step.

Specific compilation techniques for synchronous languages exist to generate efficient and reliable code for embedded controllers. Compilers produce imperative code that can be executed in a control loop triggered by external events or on a periodic signal (e.g., every millisecond). The link between logical and real time is left to the designer of the system.

Since the seminal dataflow languages Lustre (Caspi et al., 1987) and Signal (Le Guernic et al., 1991), multiple extensions of the dataflow synchronous model were proposed. Lucid Synchrone (Pouzet, 2006) combines the dataflow

```

1 node counter (init, incr: int; reset: bool) returns (n: int);
2 var pn: int;
3 let
4   pn = init -> pre n;
5   n = if reset then init else pn + incr;
6 tel
7
8 node tracker (speed, limit: int) returns (t: int);
9 var x: bool; cpt: int when x;
10 let
11   x = (speed > limit);
12   cpt = counter(0, 1, false) when x;
13   t = current(cpt);
14 tel

```

speed	28	29	32	30	44	53	58	48	33	28	29	...
limit	30	30	30	30	55	55	55	30	30	30	30	...
<i>x</i>	F	F	T	F	F	F	T	T	T	F	F	...
<i>cpt</i>				1			2	3	4			...
<i>t</i>	0	0	1	1	1	1	2	3	4	4	4	...

Stream Processing Languages and Abstractions, Fig. 2 Lustre code example with a possible execution

synchronous approach with functional features à la ML, the n-synchronous model (Mandel et al., 2010) relaxes the synchronous hypothesis by allowing communication with bounded buffers, and Zélus (Bourke and Pouzet, 2013) is a Lustre-like language extended with ordinary differential equations to define continuous-time dynamics.

Recent efforts focus on the compilation, verification, and test of dataflow synchronous programs. New techniques have been proposed to compile Lustre programs for many-core systems (Rihani et al., 2016) or improve the computation of the worst-case execution time (WCET) of the compiled code (Bon enfant et al., 2017; Forget et al., 2017). Kind2 (Champion et al., 2016) is a verification tool based on SMT solvers to model-check Lustre programs, and the Vélus compiler (Bourke et al., 2017) tackles the problem of verifying the compiler itself using a proof assistant. Lutin (Raymond and Jahier, 2013) (and its industrial counterpart, the Argosim Stimulus tool Argosim, 2015) is a DSL to design non-deterministic test scenarios for Lustre programs.

Big-Data Streaming in SPL

Big-data streaming languages are designed to handle high-throughput streams while at the same time being expressive enough to handle diverse data formats and streaming operators. A popular

way to address the requirement of high throughput is to make it easy to execute the streaming application not just on a single core or even a single computer, but on a cluster of computers. And a popular way to address the requirement of high expressiveness is to make it easy for programmers to define new streaming operators, possibly using a different programming language than the stream processing language they use for composing operators into a graph.

SPL is a big-data streaming language designed for distribution and extensibility (Hirzel et al., 2017). It was invented in 2009 and is being actively used in industry (IBM, 2008). An SPL program is an explicit stream graph of streams and operators. Unlike dataflow synchronous languages, and like other big-data streaming languages, SPL uses only minimal synchronization: an operator can fire whenever there is data available on any of its input ports, following semantics formalized in the Brooklet calculus (Soulé et al., 2010). Since synchronization across different cores and computers is hard to do efficiently, reducing synchronization simplifies distribution, giving the runtime system more flexibility for which operators to co-locate in the same core or computer. There is no assumption of simultaneity between different operator firings. When downstream operators cannot keep up with the data

```

1 stream<float64 len, rstring caller> Calls = CallsSource() { }
2
3 stream<float64 len, int32 num, rstring who> CallStats = Aggregate(Calls) {
4   window Calls: sliding, time(24.0 * 60.0 * 60.0), time(60.0);
5   output CallStats: len = Max(Calls.len),
6                     num = MaxCount(Calls.len),
7                     who = ArgMax(Calls.len, Calls.caller);
8 }
```

Stream Processing Languages and Abstractions, Fig. 3 SPL code example

rate, they implicitly throttle upstream operators via back-pressure.

Figure 3 shows an example SPL program. Line 1 defines a stream *Calls* as the output of invoking an operator *CallsSource*. In SPL, streams carry tuples that are strongly and statically typed and whose fields can hold simple numbers or strings as in the example but can also hold nested lists and tuples. The *CallsSource* operator has no further configuration (empty curly braces); for this example, assume it is user-defined elsewhere. Programmers can define their own operators either in SPL or in other languages such as C++ or Java. Lines 3–8 define a stream *CallStats* by invoking an operator *Aggregate*. The code configures the operator with an input stream *Calls*, with a **window** clause for a 24-h sliding window with 1-minute granularity, and with an **output** clause. While many operators support these and other clauses, they can also contain code restricted to the operator at hand. The *Aggregate* operator in SPL’s standard library supports intrinsic functions for *Max*, *MaxCount*, and various other aggregations. Programmers can extend SPL with new operators that, like *Aggregate*, support various configurations and operator-specific intrinsic functions.

SPL was influenced by earlier big-data streaming systems such as Borealis (Abadi et al., 2005) and TelegraphCQ (Chandrasekaran et al., 2003), generalizing them to be less dependent on relational data and more extensible. Various other streaming systems after SPL, such as Storm (Toshniwal et al., 2014) and Spark Streaming (Zaharia et al., 2013), also target big-data streaming. Like SPL, they use explicit stream graphs as their core abstraction, but unlike SPL, they use embedded (not stand-alone) domain-specific languages.

While the examples of Lustre and SPL draw a stark contrast between synchronous dataflow and big-data streaming, there are also commonalities. For instance, the StreamIt language is synchronous, but like big-data streaming languages, it uses an explicit stream graph as its core abstraction (Thies et al., 2002). And Soulé et al. show how to reduce the dependence of StreamIt on synchrony (Soulé et al., 2013).

Future Directions for Research

The landscape of streaming languages is far from consolidating on any dominant approach. New languages keep coming out to address a variety of open issues. One active area of research is the interaction between streams (data in motion) and state (data at rest). While CQL gave a conceptually clean answer (Arasu et al., 2006), people are debating alternative approaches, such as the Lambda architecture (Marz, 2011) and the Kappa architecture (Kreps, 2014). Another active area of research is how to handle uncertainty, such as out-of-order data, missing fields, erroneous sensor readings, approximate algorithms, or faults. On this front, streaming languages have not yet reached the clarity of databases with their ACID properties. When it comes to implementation strategies, there has been a recent surge in embedded domain-specific languages. But while EDSLs have fewer tooling needs and are less intimidating for users familiar with their host language, they are less self-contained and offer less static optimization and error-checking than stand-alone languages. We hope this article inspires innovation in streaming languages that are well-informed by those that came before.

Cross-References

- Continuous Queries
- Languages for Big Data Analysis

References

- Abadi DJ, Ahmad Y, Balazinska M, Cetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey AS, Rasin A, Ryvkina E, Tatbul N, Xing Y, Zdonik S (2005) The design of the Borealis stream processing engine. In: Conference on innovative data systems research (CIDR), pp 277–289
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. *J Very Large Data Bases (VLDB J)* 15(2):121–142
- Argosim (2015) Stimulus. <http://argosim.com/>. Retrieved Nov 2017
- Arnold M, Grove D, Herta B, Hind M, Hirzel M, Iyengar A, Mandel L, Saraswat V, Shinnar A, Siméon J, Takeuchi M, Tardieu O, Zhang W (2016) META: middleware for events, transactions, and analytics. *IBM J Res Dev* 60(2–3):15:1–15:10
- Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Symposium on principles of database systems (PODS), pp 1–16
- Barbieri DF, Braga D, Ceri S, Della Valle E, Grossniklaus M (2009) C-SPARQL: SPARQL for continuous querying. In: Poster at international World Wide Web conferences (WWW-poster), pp 1061–1062
- Benveniste A, Caspi P, Edwards SA, Halbwachs N, Le Guernic P, De Simone R (2003) The synchronous languages 12 years later. *Proc IEEE* 91(1):64–83
- Bonenfant A, Carrier F, Cassé H, Cuenot P, Claraz D, Halbwachs N, Li H, Maiza C, De Michiel M, Mussot V, Parent-Vigouroux C, Puaut I, Raymond P, Rohou E, Sotin P (2016) When the worst-case execution time estimation gains from the application semantics. In: European congress on embedded real-time software and systems (ERTS2)
- Bourke T, Pouzet M (2013) Zélus: a synchronous language with odes. In: Conference on hybrid systems: computation and control (HSCC), pp 113–118
- Bourke T, Brun L, Dagand P, Leroy X, Pouzet M, Rieg L (2017) A formally verified compiler for Lustre. In: Conference on programming language design and implementation (PLDI), pp 586–601
- Caspi P, Pilaud D, Halbwachs N, Plaice JA (1987) LUSTRE: a declarative language for real-time programming. In: Symposium on principles of programming languages (POPL), pp 178–188
- Champion A, Mebsout A, Sticksel C, Tinelli C (2016) The Kind 2 model checker. In: Conference on computer aided verification (CAV), pp 510–517
- Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein JM, Hong W, Krishnamurthy S, Madden S, Raman V, Reiss F, Shah MA (2003) TelegraphCQ: continuous dataflow processing for an uncertain world. In: Conference on innovative data systems research (CIDR)
- Chen J, DeWitt DJ, Tian F, Wang Y (2000) NiagaraCQ: a scalable continuous query system for internet databases. In: International conference on management of data (SIGMOD), pp 379–390
- Colaco JL, Pagano B, Pouzet M (2017) Scade 6: a formal language for embedded critical software development. In: International symposium on theoretical aspect of software engineering (TASE), pp 1–10
- Diao Y, Fischer PM, Franklin MJ, To R (2002) YFilter: efficient and scalable filtering of XML documents. In: Demo at international conference on data engineering (ICDE-Demo), pp 341–342
- Forget J, Boniol F, Pagetti C (2017) Verifying end-to-end real-time constraints on multi-periodic models. In: International conference on emerging technologies and factory automation (ETFA)
- Hirzel M (2012) Partition and compose: parallel complex event processing. In: Conference on distributed event-based systems (DEBS), pp 191–200
- Hirzel M, Rabah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60
- Hirzel M, Schneider S, Gedik B (2017) SPL: an extensible language for distributed stream processing. *Trans Program Lang Syst (TOPLAS)* 39(1):5:1–5:39
- Hudak P (1998) Modular domain specific languages and tools. In: International conference on software reuse (ICSR), pp 134–142
- IBM (2008) IBM streams. <https://ibmstreams.github.io>. Retrieved Nov 2017
- Jain N, Mishra S, Srinivasan A, Gehrke J, Widom J, Balakrishnan H, Cetintemel U, Cherniack M, Tibbets R, Zdonik S (2008) Towards a streaming SQL standard. In: Conference on very large data bases (VLDB), pp 1379–1390
- Johnston WM, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages. *ACM Comput Surv (CSUR)* 36(1):1–34
- Kreps J (2014) Questioning the Lambda architecture. <http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html>. Retrieved Nov 2017
- Kuhn T (2014) A survey and classification of controlled natural languages. *Comput Linguist* 40(1):121–170
- Le Guernic P, Gautier T, Le Borgne M, Le Maire C (1991) Programming real-time applications with SIGNAL. *Proc IEEE* 79(9):1321–1336
- Mandel L, Plateau F, Pouzet M (2010) Lucy-n: a n-synchronous extension of Lustre. In: International conference on mathematics of program construction (MPC), pp 288–309
- Marz N (2011) How to beat the CAP theorem. <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>. Retrieved Nov 2017
- Mendell M, Nasgaard H, Bouillet E, Hirzel M, Gedik B (2012) Extending a general-purpose streaming system

- for XML. In: Conference on extending database technology (EDBT), pp 534–539
- Pouzet M (2006) Lucid synchrone, version 3, Tutorial and reference manual
- Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. <http://www.w3.org/TR/rdf-sparql-query/>. Retrieved Nov 2017
- Raymond P, Jahier E (2013) Lutin reference manual version Trilby-1.54
- Rihani H, Moy M, Maiza C, Davis RI, Altmeyer S (2016) Response time analysis of synchronous data flow programs on a many-core processor. In: Conference on real-time networks and systems (RTNS), pp 67–76
- Shinnar A, Siméon J, Hirzel M (2015) A pattern calculus for rule languages: expressiveness, compilation, and mechanization. In: European conference on object-oriented programming (ECOOP), pp 542–567
- Soulé R, Hirzel M, Grimm R, Gedik B, Andrade H, Kumar V, Wu KL (2010) A universal calculus for stream processing languages. In: European symposium on programming (ESOP), pp 507–528
- Soulé R, Gordon MI, Amarasinghe S, Grimm R, Hirzel M (2013) Dynamic expressivity with static optimization for streaming languages. In: Conference on distributed event-based systems (DEBS), pp 159–170
- Soulé R, Hirzel M, Gedik B, Grimm R (2016) River: an intermediate language for stream processing. Softw Pract Exp (SP&E) 46(7):891–929
- Stephens R (1997) A survey of stream processing. Acta Informatica 34(7):491–541
- Thies W, Karczmarek M, Gordon M, Maze D, Wong J, Hoffmann H, Brown M, Amarasinghe S (2002) StreamIt: a compiler for streaming applications. Technical report LCS-TM-622, MIT
- Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm @Twitter. In: International conference on management of data (SIGMOD), pp 147–156
- Vaziri M, Tardieu O, Rabbah R, Suter P, Hirzel M (2014) Stream processing with a spreadsheet. In: European conference on object-oriented programming (ECOOP), pp 360–384
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Symposium on operating systems principles (SOSP), pp 423–438
- Zemke F, Witkowski A, Cherniak M, Colby L (2007) Pattern matching in sequences of rows. Technical report, ANSI Standard Proposal

Stream Processing Optimization

► Stream Query Optimization

Stream Query Optimization

Martin Hirzel¹, Robert Soulé², Buğra Gedik³, and Scott Schneider¹

¹IBM Research AI, Yorktown Heights, NY, USA

²Università della Svizzera Italiana (USI), Lugano, Switzerland

³Department of Computer Engineering, Bilkent University, Ankara, Turkey

Synonyms

Continuous query optimization; Stream processing optimization

Definitions

Stream query optimization is the process of modifying a stream processing query, often by changing its graph topology and/or operators, with the aim of achieving better performance (such as higher throughput, lower latency, or reduced resource usage), while preserving the semantics of the original query.

Overview

A stream query optimization modifies a stream query to make it faster. Users want stream queries to be fast for several reasons. They want to grasp opportunities or avert risks observable on the input streams before it is too late. They want any views derived from the input streams to be up-to-date and not stale. And they want their system to keep up with the rate of input streams without falling behind, which would require shedding load or saving data to disk for later processing.

Knowing about stream query optimizations helps developers at all layers. Application developers who know about stream query optimizations can get the most out of the optimizations built into their streaming platform and can supplement them by hand-optimizing their application where necessary. Streaming platform developers can use knowledge about

stream query optimizations to make their platform faster by implementing additional optimizations or by generalizing their existing optimizations to apply in more situations. Finally, researchers who invent new optimizations need to know the state-of-the-art optimizations to channel their efforts into the most innovative and impactful direction.

The rest of this section introduces some basic concepts and gives a high-level overview and categorization of the most common stream query optimizations.

An optimization should be both safe and profitable. An optimization is *safe* if it can be applied to a stream query without changing what it computes, as determined by the user's requirements. An optimization is *profitable* if it makes the stream query faster, as measured by metrics that matter to the user, such as throughput, latency, or resource efficiency. There is a substantial literature on different stream query optimizations, with different safety and profitability characteristics. This entry lists the most common optimizations along with short descriptions. More in-depth descriptions can be found in our survey paper and tutorial on stream processing optimizations (Hirzel et al., 2014; Schneider et al., 2013).

Stream query optimizations are best understood with respect to stream graphs. A *stream graph* is a directed graph whose edges are streams and whose nodes are operators. Root and leaf nodes are called sources and sinks, respectively.

This entry uses terminology that makes only few assumptions so as not to unnecessarily restrict its scope. For instance, this entry does not assume restrictions on the shape of the stream graph: unless specified otherwise, it does not assume that stream graphs are acyclic, or are single-source-single-sink, or are trees. A *stream* is an ordered sequence of data items, which are values that can range from simple numbers to flat tuples to more elaborate structured data that may be deeply nested and have variable size. Streams are conceptually infinite, in the sense that as the streaming computation unfolds over time, the sequence of data items is unbounded in length. *Operators* are primarily stream transformers but can also have state and side effects beyond the output streams they produce. Indeed, sources and sinks are operators that typically have the side effect of continuously consuming input from and producing output to the external world outside of the stream graph.

Figure 1 lists the most popular stream query optimizations. Each optimization has a symbol (e.g., **B**), a name (e.g., Batching), and two annotations indicating its effect on the graph and the semantics. The optimizations to the left (shown in orange and red) leave the stream graph unchanged. This means the graph still contains the same nodes and edges, and any changes are limited to the behavior of individual operators. The optimizations on the right (shown in blue and green) change the stream graph. The optimizations on the top (shown in orange and blue)

B 1 Batching <small>Unchanged graph Stable semantics</small>	Fu 2 Fusion <small>Changed graph Stable semantics</small>
P 3 Placement <small>Unchanged graph Stable semantics</small>	Os 5 Operator separation <small>Changed graph Stable semantics</small>
Ss 4 State sharing <small>Unchanged graph Stable semantics</small>	Or 6 Operator reordering <small>Changed graph Stable semantics</small>
Lb 8 Load balancing <small>Unchanged graph Stable semantics</small>	Re 7 Redundancy elimination <small>Changed graph Stable semantics</small>
As 9 Algorithm selection <small>Unchanged graph Unstable semantics</small>	Fi 11 Fission <small>Changed graph Unstable semantics</small>
Ls 10 Load shedding <small>Unchanged graph Unstable semantics</small>	

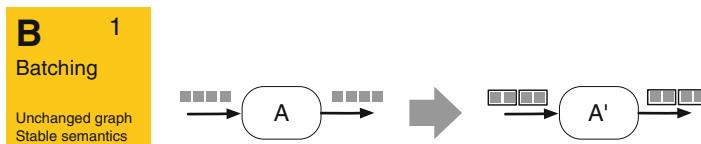
Stream Query Optimization, Fig. 1 Overview of stream query optimizations discussed in this entry

keep the semantics stable. This means that as long as their safety preconditions are satisfied, the observable behavior of the stream query is the same before and after applying the optimization. These optimizations are safe in the sense discussed above. The extreme case is batching and fusion at the very top, which are not only safe, but have safety preconditions that are trivial to satisfy. Only the three optimizations at the very

bottom have unstable semantics: load shedding, which always changes the semantics, and algorithm selection and fission, which sometimes change semantics if the algorithm is approximate or if the fission perturbs the order of data items. These forms of semantic changes are sometimes tolerable depending on application requirements.

The following section will elaborate on each of the optimizations from Fig. 1 with illustrations, definitions, and literature references.

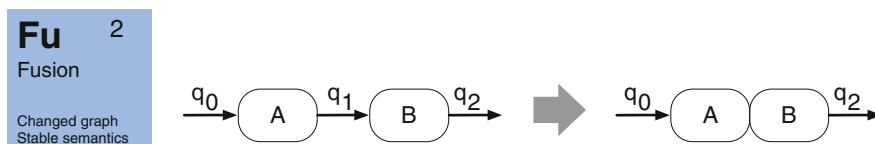
Key Research Findings



Batching Definition: Batching reduces overhead by processing multiple data items together. This optimization improves throughput by amortizing the cost of operator-firing and communication over several data items. However, this throughput gain is usually at the expense of additional latency, as an operator cannot fire until

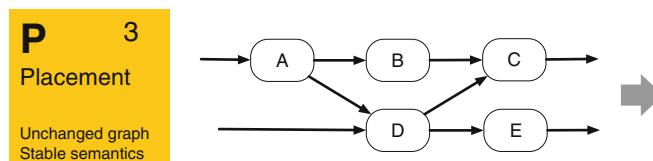
it has received a batch-size number of data items.

References: In the literature, batching is also called train scheduling (Carney et al., 2003) and execution scaling (Gordon et al., 2006). Batching can be dynamic, as in SEDA (Welsh et al., 2001), or static, as in StreamIt (Gordon et al., 2006).



Fusion Definition: Fusion combines smaller operators into a larger one, to avoid the overhead of data serialization and transport. Operators may be fused in many ways, for example, by placing the operators into the same thread or by keeping operators in separate threads that share a common address space. Fusion may come at the cost

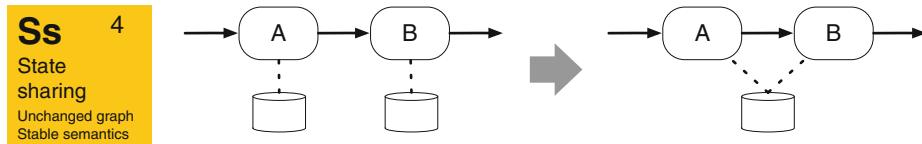
of decreased pipeline parallelism. **References:** StreamIt (Gordon et al., 2002) aggressively fuses fine-grained operators, followed by fission. In Aurora, fusion is referred to as superbox scheduling (Tatbul et al., 2003). System S uses the COLA fusion optimizer (Khandekar et al., 2009), which balances safety and profitability constraints.



Placement Definition: Placement assigns operators to hosts and cores to reduce communication costs or better utilize available resources. Frequently, these two goals are at odds. When

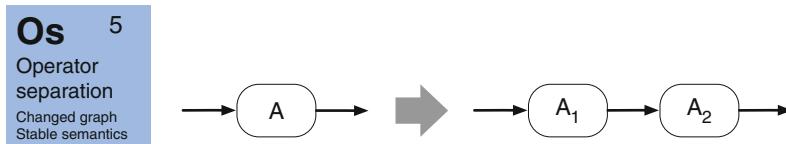
multiple operators are placed on the same host, they communicate at lower cost, but they compete for common resources, such as disk, memory, or CPU. On the other hand, when operators

are placed on different hosts, that reduces contention but incurs higher communication costs. *References:* StreamIt uses placement to optimize streaming applications deployed on multi-core machines with nonuniform memory access (Gordon et al., 2002). Pietzuch et al. use metrics gathered from network conditions to place operators in a distributed setting (Pietzuch et al., 2006). SODA incorporates job admission with the placement decisions (Wolf et al., 2008).



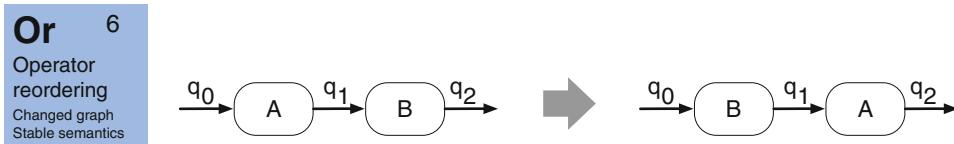
State sharing *Definition:* State sharing attempts to avoid unnecessary copies of data. While the main goal of the optimization is to reduce the memory footprint of a streaming application, it can also impact performance by reducing stalls due to cache misses or disk I/O. *References:* State sharing can be applied generally between

streaming operators, such as in the work of Brito et al. (2008). Or, it can be applied in more restricted forms, such as in CQL, which shares only window state (Arasu et al., 2006), or in the work of Sermulins et al., which shares queue state between two pipelined operators (Sermulins et al., 2005).



Operator separation *Definition:* Operator separation splits a large computation into smaller steps. In some cases, this optimization can result in reduced resource consumption. Often, this optimization is used to enable other optimizations, such as operator reordering. *References:* Using algebraic equivalences for separating operators is

a common technique in database query execution planning (Garcia-Molina et al., 2008). Yu et al. (2009) present a stream query compiler that uses explicit annotations to determine when to separate aggregate operators. Decoupled software pipelining separates general code by analyzing data dependencies from first principles (Ottoni et al., 2005).

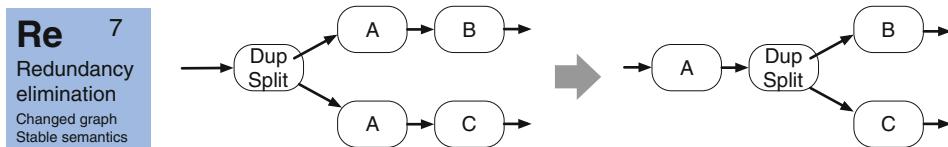


Operator reordering *Definition:* A reordering optimization moves more selective operators, which reduce the data volume, upstream. This has the benefit of reducing the data flowing into downstream computation, thus eliminating unnecessary work. However, care must be taken

to preserve the desired semantics, and operators should only be re-ordered if the operations are commutative. *References:* Reordering based on the properties of relational algebra is common in database query planning (Garcia-Molina et al., 2008). The Volcano (Graefe, 1990) system

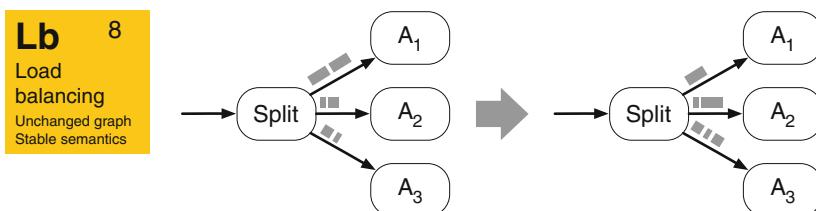
implements a particularly profitable case of reordering: swapping split and merge operators in a data-parallel pipeline to avoid choke-points.

Eddies (Avnur et al., 2000) is a dynamic technique for finding the most profitable ordering of operators with independent selectivities.



Redundancy elimination *Definition:* Redundancy elimination eliminates superfluous computations. This optimization must ensure that removing a given computation does not change the resulting output. While, in general, determining program equivalence is undecidable, in practice, streaming languages are often based on an algebra, which makes the optimization

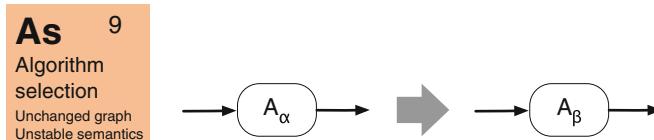
feasible. *References:* The Rete algorithm (Forgy, 1982) is a highly-influential approach to detecting and eliminating redundancies in a multi-tenant system. NiagaraCQ (Chen et al., 2000) applied similar ideas to processing streaming XML. Pietzuch et al. (2006) eliminate redundancy at application launch time.



Load balancing *Definition:* Load balancing attempts to distribute workload evenly across resources. To be effective, a load-balancing optimization must adapt to workload skew, e.g., when there are many accesses to a popular data item. *References:* River uses intelligent routing for load balancing in a cluster (Arpac-Dusseau et al., 1999). Caneill et al. use online

adaptive routing, which considers downstream communication (Caneill et al., 2016). In contrast, Amini et al. use operator placement for load balancing in a cluster (Amini et al., 2006). StreamIt also uses placement for load balancing, but targets multi-core machines (Gordon et al., 2002).

S

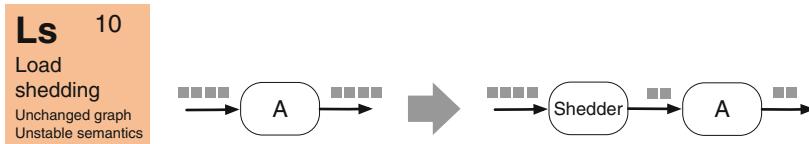


Algorithm selection *Definition:* Algorithm selection uses a different algorithm to implement an operator. There are various reasons to change the algorithm. For example, one algorithm may be more general than another. One algorithm may

optimize for different criteria. Or, one algorithm may perform better than others under different circumstances. *References:* Changing the operator algorithm for particular workloads is common in database systems (e.g., using a hash join vs.

a nested loop join Garcia-Molina et al., 2008). In streaming systems, SEDA allows an operator to pick a different algorithm to provide degraded service (Welsh et al., 2001). Borealis allows an

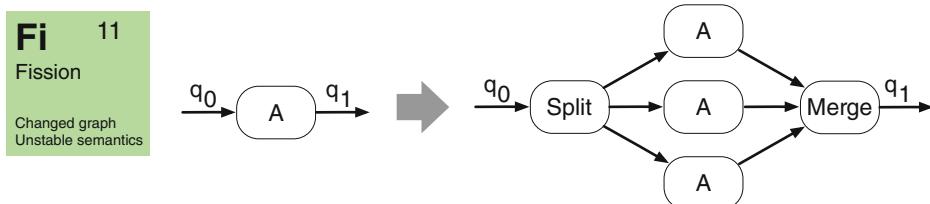
operator to switch to a different algorithm based on a control input (Abadi et al., 2005). And, SODA offers algorithm selection at the granularity of entire jobs (Wolf et al., 2008).



Load shedding Definition: Load shedding copes with high load by dropping data items to process. Load shedding may change the expected results of a computation, and therefore the semantics of the streaming application.

References: Aurora implements priority-based

load shedding (Tatbul et al., 2003). Compact Shedding Filters perform load-shedding at data-generating sensors, rather than the server, to avoid unnecessary network communication (Gedik et al., 2008).



Fission Definition: Fission, often referred to as data parallelism, attempts to process multiple data items in parallel by replicating an operator. Note that when parallelizing operators, the optimizer must respect ordering or state constraints to ensure the correctness of the computation. **References:** The StreamIt compiler applies fission to stateless operators with static selectivity (Gordon et al., 2006). Schneider et al. explored how to make fission safe in the more general case of partitioned-stateful and selective operators (Schneider et al., 2015). Finally, Brito et al. propose using transactional memory to make fission safe in the case of arbitrary operator state (Brito et al., 2008).

Fission One application that benefits a lot from fission is streaming radio astronomy, which forms evolving imaging maps of radio emission from the sky over a wide range of frequencies (Biem et al., 2010b). High-performance processing is a critical requirement in this application, due to the sheer volume of sensor data that flow in real-time from a very large array of antennas, which is typical of phased array radio telescopes (see SKA 2000). The application's flow graph is organized as a split-merge topology. An initial operator performs frequency mapping and blocking so that a subsequent operator splits the data into blocks of frequency channels. Each such block is then processed in parallel by performing indexing and convolution operations, forming a costly stateless parallel region that highly benefits from fission. Finally, the results are merged via an aggregation to form the complete imaging map.

Fusion In Biem et al. (2010a), a streaming application is presented for maintaining live traffic

Examples of Application

We illustrate the use of the three most popular optimizations, namely, fission, fusion, and batching, in the context of real-world streaming applications from the literature.

status information using GPS sensor data. The application processes floating car data originating from public transportation vehicles to extract up-to-date traffic information, such as speed and traffic flow measurements at the level of streets within a city, traffic volume measurements by region, estimates of travel times between different points, etc. The application is organized as a graph of streams and operators, where different operators perform individual tasks, such as data parsing and cleaning, snapping GPS points to roads, aggregation and statistics maintenance, prediction of travel times, etc. The operators in the flow graph of the application are grouped into processing elements, which are then distributed across machines. The operators that are assigned to the same processing element are fused inside a shared address space, in order to reduce the data transfer latency.

Batching The batching optimization is particularly useful when interacting with external systems. A common use case for such interaction is managing state. For instance, LinkedIn, which is a business- and employment-oriented social networking service, runs several streaming applications that manage state, such as user profiles and aggregate counts. These applications include email digest generation, top-k relevant category detection, and profile update standardization, among others (Noghabi et al., 2017). In these applications, the state needs to be accessed from either the local disk-based or remote state management systems. Similarly, many of these streaming applications write their output to an external system, such as a message queue. When an external interaction is to be performed on a per-tuple basis, batching is an effective optimization that can amortize the overheads and significantly improve the performance.

Future Directions for Research

In an ideal world, programmers would code their streaming applications at the most natural level of abstraction without having to worry about runtime performance, and the streaming system

would automatically execute the applications with consistently high performance. While stream query optimizations have made significant advances toward this goal, they still fall short on automation and predictability.

It is still difficult to fully automate streaming optimizations, because automatic optimizers may not find the most profitable setting. Finding the most profitable setting ahead-of-time, before executing the streaming application, is tricky because the performance model may be complicated (e.g., when multiple optimizations interact) or some information required by the performance model may be missing or hard to predict. An alternative to ahead-of-time optimization is online feedback-directed optimization, which is subject to the trade-offs inherent in the SASO properties of feedback control (stability, accuracy, settling, and no overshoot) (Hellerstein et al., 2004). Work such as that of De Matteis and Mencagli, which uses a control-theoretic approach to perform online adaptations to optimize for latency (De Matteis et al., 2016), is a promising direction.

When streaming applications fall short of their expected peak performance, programmers often optimize them by hand. This has the advantage of giving programmers more control over squeezing the last bit of performance out of their application. Unfortunately, it can also clutter their code, can make the performance brittle by over-fitting to the current workload and execution environment, and may even inhibit automated optimizations. One approach to address these issues is to decouple the optimizer hints and directives from the core application logic (Hirzel et al., 2017).

Overall, there is still much work to be done in developing more effective optimizations, more reliable performance models, and more unintrusive language features for giving optimizer hints. This entry represents a snapshot of the state of the art, and we encourage the reader to venture beyond it.

Cross-References

- ▶ [Continuous Queries](#)
- ▶ [Introduction to Stream Processing Algorithms](#)
- ▶ [Sliding-Window Aggregation Algorithms](#)

References

- Abadi DJ, Ahmad Y, Balazinska M, Çetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey AS, Rasin A, Ryvkina E, Tatbul N, Xing Y, Zdonik S (2005) The design of the Borealis stream processing engine. In: Conference on innovative data systems research (CIDR), pp 277–289
- Amini L, Jain N, Sehgal A, Silber J, Verscheure O (2006) Adaptive control of extreme-scale stream processing systems. In: International conference on distributed computing systems (ICDCS)
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. *J Very Large Data Bases (VLDB J)* 15(2): 121–142
- Arpacı-Dusseau RH, Anderson E, Treuhaft N, Culler DE, Hellerstein JM, Patterson D, Yelick K (1999) Cluster I/O with river: making the fast case common. In: Workshop on I/O in parallel and distributed systems (IOPADS), pp 10–22
- Avnur R, Hellerstein JM (2000) Eddies: continuously adaptive query processing. In: International conference on management of data (SIGMOD), pp 261–272
- Biem A, Bouillet E, Feng H, Ranganathan A, Riabov A, Verscheure O, Koutsopoulos HN, Rahmani M, Guc B (2010a) Real-time traffic information management using stream computing. *IEEE Data Eng Bull* 33(2): 64–68
- Biem A, Elmegreen B, Verscheure O, Turaga D, Andrade H, Cornwell T (2010b) A streaming approach to radio astronomy imaging. In: Conference on acoustics, speech, and signal processing (ICASSP), pp 1654–1657
- Brito A, Fetzer C, Sturzheim H, Felber P (2008) Speculative out-of-order event processing with software transaction memory. In: Conference on distributed event-based systems (DEBS), pp 265–275
- Caneill M, El Rhedane A, Leroy V, De Palma N (2016) Locality-aware routing in stateful streaming applications. In: International conference on middleware, pp 4:1–4:13
- Carney D, Çetintemel U, Rasin A, Zdonik S, Cherniack M, Stonebraker M (2003) Operator scheduling in a data stream manager. In: Conference on very large data bases (VLDB), pp 309–320
- Chen J, DeWitt DJ, Tian F, Wang Y (2000) NiagaraCQ: a scalable continuous query system for internet databases. In: International conference on management of data (SIGMOD), pp 379–390
- De Matteis T, Mencagli G (2016) Keep calm and react with foresight: strategies for low-latency and energy-efficient elastic data stream processing. In: Principles and practice of parallel programming (PPoPP), pp 13:1–13:12
- Forgy CL (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artif Intell* 19:17–37
- Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Prentice Hall, Upper Saddle River
- Gedik B, Wu KL, Yu PS (2008) Efficient construction of compact shedding filters for data stream processing. In: International conference on data engineering (ICDE), pp 396–405
- Gordon MI, Thies W, Karczmarek M, Lin J, Meli AS, Lamb AA, Leger C, Wong J, Hoffmann H, Maze D, Amarasinghe S (2002) A stream compiler for communication-exposed architectures. In: Conference on architectural support for programming languages and operating systems (ASPLOS), pp 291–303
- Gordon MI, Thies W, Amarasinghe S (2006) Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In: Conference on architectural support for programming languages and operating systems (ASPLOS), pp 151–162
- Graefe G (1990) Encapsulation of parallelism in the Volcano query processing system. In: International conference on management of data (SIGMOD), pp 102–111
- Hellerstein JL, Diao Y, Parekh S, Tilbury DM (2004) Feedback control of computing systems. Wiley, Hoboken
- Hirzel M, Soulé R, Schneider S, Gedik B (2014) A catalog of stream processing optimizations. *ACM Comput Surv (CSUR)* 46(4):1–34
- Hirzel M, Schneider S, Gedik B (2017) SPL: an extensible language for distributed stream processing. *Trans Program Lang Syst (TOPLAS)* 39(1):5:1–5:39
- Khandekar R, Hildrum I, Parekh S, Rajan D, Wolf J, Wu KL, Andrade H, Gedik B (2009) COLA: optimizing stream processing applications via graph partitioning. In: International conference on middleware, pp 308–327
- Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza: stateful scalable stream processing at LinkedIn. In: Conference on very large data bases (VLDB), pp 1634–1645
- Otonni G, Rangan R, Stoler A, August DI (2005) Automatic thread extraction with decoupled software pipelining. In: International symposium on microarchitecture (MICRO), pp 105–118
- Pietzuch P, Ledlie J, Schneidman J, Roussopoulos M, Welsh M, Seltzer M (2006) Network-aware operator placement for stream-processing systems. In: International conference on data engineering (ICDE), pp 49–61
- Schneider S, Gedik B, Hirzel M (2013) Tutorial: stream processing optimizations. In: Conference on distributed event-based systems (DEBS), pp 249–258
- Schneider S, Hirzel M, Gedik B, Wu KL (2015) Safe data parallelism for general streaming. *IEEE Trans Comput (TC)* 64(2):504–517
- Sermulins J, Thies W, Rabba R, Amarasinghe S (2005) Cache aware optimization of stream programs. In:

- Conference on languages, compiler, and tool support for embedded systems (LCTES), pp 115–126
- SKA Telescope (2000) Square kilometre array telescope. <https://skatelescope.org>. Retrieved Nov 2017
- Tatbul N, Cetintemel U, Zdonik S, Cherniack M, Stonebraker M (2003) Load shedding in a data stream manager. In: Conference on very large data bases (VLDB), pp 309–320
- Welsh M, Culler D, Brewer E (2001) SEDA An architecture for well-conditioned, scalable Internet services. In: Symposium on operating systems principles (SOSP), pp 230–243
- Wolf J, Bansal N, Hildrum K, Parekh S, Rajan D, Wagle R, Wu KL, Fleischer L (2008) SODA: an optimizing scheduler for large-scale stream-based distributed computer systems. In: International conference on middleware, pp 306–325
- Yu Y, Gunda PK, Isard M (2009) Distributed aggregation for data-parallel computing: interfaces and implementations. In: Symposium on operating systems principles (SOSP), pp 247–260

Stream Reasoning

- ▶ Semantic Stream Processing

Stream Reduce

- ▶ Sliding-Window Aggregation Algorithms

Stream Window Aggregation Semantics and Optimization

Paris Carbone¹, Asterios Katsifodimos², and Seif Haridi¹

¹KTH Royal Institute of Technology,
Stockholm, Sweden

²TU Delft, Delft, Netherlands

Definitions

Sliding windows are bounded sets which evolve together with an infinite data stream of records. Each new sliding window evicts records from the previous one while introducing newly

arrived records as well. Aggregations on windows typically derive some metric such as an average or a sum of a value in each window. The main challenge of applying aggregations to sliding windows is that a naive execution can lead to a high degree of redundant computation due to a large number of common records across different windows. Special optimization techniques have been developed throughout the years to tackle redundancy and make sliding window aggregation feasible and more efficient in large data streams.

Overview

Data stream processing has evolved significantly throughout the years, both in terms of system support and in programming model primitives. Alongside adopting common data-centric operators from relational algebra and functional programming such as select, join, flatmap, reduce, etc., stream processors introduced a new set of primitives that are exclusive to the evolving nature of unbounded data. Stream windows are perhaps the most common and widely studied primitive in stream processing which is used to express computation on continuously evolving subsets out of a possibly never-ending stream. In essence, stream windows grant control on the granularity and the scope of stream aggregations.

Several early stream processing systems (e.g., TelegraphCQ (Chandrasekaran et al., 2003), STREAM (Arasu et al., 2004)) provided support for windowing through a predefined set of primitives to construct time- and count-based sliding windows. For example, periodic tumbling and sliding windows were already standardized as early as the SQL-99 standard and studied thoroughly in the Continuous Query Language (CQL) (Arasu et al., 2006) as well as the stream processing language (SPL) (Hirzel et al., 2009) among others. A *tumbling* window is a simple case of a stream window type, which is defined as a sequence of periodic consecutive sets of records in a stream with a fixed length that is termed *range*. For example, if we assume a stream of car speed events, the following simple query in CQL would discretize that stream into

windows of every 30sec interval and compute the maximum speed per window:

```
SELECT max(speed)
from CarEvents
[RANGE 30 Seconds]
```

In principle, in tumbling windows each record can only belong to a single window. As a result, the evaluation of each window can be performed trivially by grouping records by the window they belong to and executing each window aggregation independently. However, sliding windows add a challenging twist to the formula, namely, the “slide.” As an example consider the following sliding window query in SQL-99:

```
SELECT AVERAGE(speed)
FROM CarEvents
[WATTR timestamp
RANGE 7 minute
SLIDE 3 minute]
```

The *slide* represents “when” or “how often” a window has to be evaluated while including all records defined in its *range* in the computation of the average speed. In this sliding window example, there is an overlap of 5 min between each consecutive window, and thus, a naive execution would result into redundant operations in its great majority. Beyond periodic windows, today’s Apache open-source systems such as Flink (Carbone et al., 2015, 2017), Beam, and Apex provide support for more advanced, often user-defined, sliding window definitions such as session (Akidau et al., 2015) or content-driven windows (Bifet and Gavalda, 2007), among others.

Sliding windows have their own set of optimization techniques that aim to reduce the redundant computations caused by the intersection of events between neighboring windows. In this paper we categorize optimization techniques into *slicing*, *pre-aggregation*, and *hybrid* and analyze them throughout the rest of this chapter.

Basic Concepts

There are many interpretations of window semantics, from simple range and count event-time windows (Arasu et al., 2006) to policy-based (Hirzel

et al., 2009) and composite event-time windows with retraction for out-of-order processing (Li et al., 2008b). The SECRET model (Botan et al., 2010) aimed to subsume most of the windowing semantics proposed in academia and commercial systems. However, for the sake of brevity, a more simplified description is used here, with a heavy focus on window aggregation, based on the recent work on the Cutty aggregator (Carbone et al., 2016) and FlatFat (Tangwongsan et al., 2015).

Stream Discretization

Data streams are unbounded sequences of records which are described by a given schema T . More formally, a stream $\bar{s} \in \text{Seq}(T)$ is a sequence out of all possible sequences $\text{Seq}(T)$ over T . Windows are finite subsequences reflecting intervals of a stream \bar{s} . An interval $s[a, b]$ is simply a set of records from index a to b over a stream \bar{s} and the set of all possible intervals $\text{Str}(T) \subset \text{Seq}(T)$.

In their most general form, windows can be derived by *discretizing* an unbounded stream. A *Discretizer* transforms a stream $\bar{s} \in \text{Seq}(T)$ into a sequence $\bar{w} \in \text{Seq}(\text{Str}(T))$ of (possibly overlapping) windows. In the most system-agnostic manner, every possible discretization of a stream can be aided through a discretization function provided to a special *Discretize* or *Windowing* stream operator.

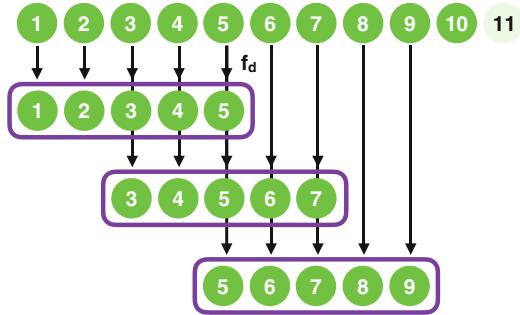
Definition 1 $\text{Discretize} : f_{\text{disc}} \times \text{Seq}(T) \rightarrow \text{Seq}(\text{Str}(T))$

Figure 1 depicts a simple example of a discretization function f_d applied on a stream of elements which forms count windows with a “range” of 5 records and a “slide” of 2 records.

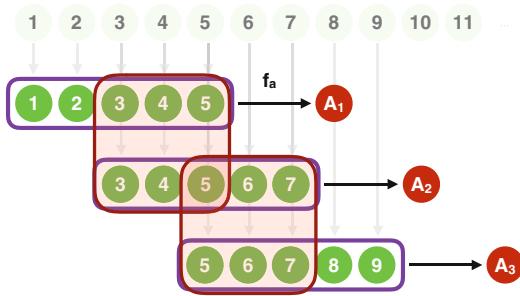
Aggregating Sliding Windows

Conceptually, in data stream programming models, an *Aggregate* operator computes an aggregation on each window derived after a stream discretization, given an aggregation function f_a .

Definition 2 $\text{Aggregate} : (f_a : \text{Str}(T) \rightarrow T') \times \text{Seq}(\text{Str}(T)) \rightarrow \text{Seq}(T')$



Stream Window Aggregation Semantics and Optimization, Fig. 1 Discretization of a count window of range 5 and slide 2



Stream Window Aggregation Semantics and Optimization, Fig. 2 Aggregation of a count window of range 5 and slide 2

Examples of f_a is a SUM or AVG, but also more complex aggregations can be executed in a window, such as building a machine learning model. Figure 2 shows how aggregates are formed on each consecutive window of a discretized stream. The main reason for optimizing the window aggregation process stems from two issues that can be observed in this example. First, a consecutive execution of the aggregation operation after discretization can be inefficient both in terms of space needed to log all elements of each window as windows can be very large in size. At the same time, the response time has to be minimized when iterating through all window contents in order to calculate an aggregate. Finally, and most importantly, as highlighted in Fig. 2, sliding windows might involve a large amount of overlapping across consecutive windows. In this example there is an overlapping of three records between every two windows.

The first problem is solved by simply pipelining discretization with aggregation and thus, effectively providing a partial evaluation of window aggregates. Partial aggregation is described in section “[Partial Window Aggregation](#)”. The overlapping problem is more complex, and its optimization techniques are further classified by window types into slicing, general pre-aggregation, and hybrid techniques, covered thoroughly in sections “[Window Slicing](#)”, “[General Pre-aggregation](#)”, and “[Cutty: A Hybrid Approach](#)”, respectively.

Partial Window Aggregation

A complete partial window aggregation scheme has been proposed in both FlatFat (Tangwongsan et al., 2015) and Cutty (Carbone et al., 2016). According to that scheme, an aggregation f_a can be decomposed into partial aggregation operations in order to pipeline the process during discretization. A window aggregation function is therefore decomposed into `lift` and `lower` and a `combine` functions as such:

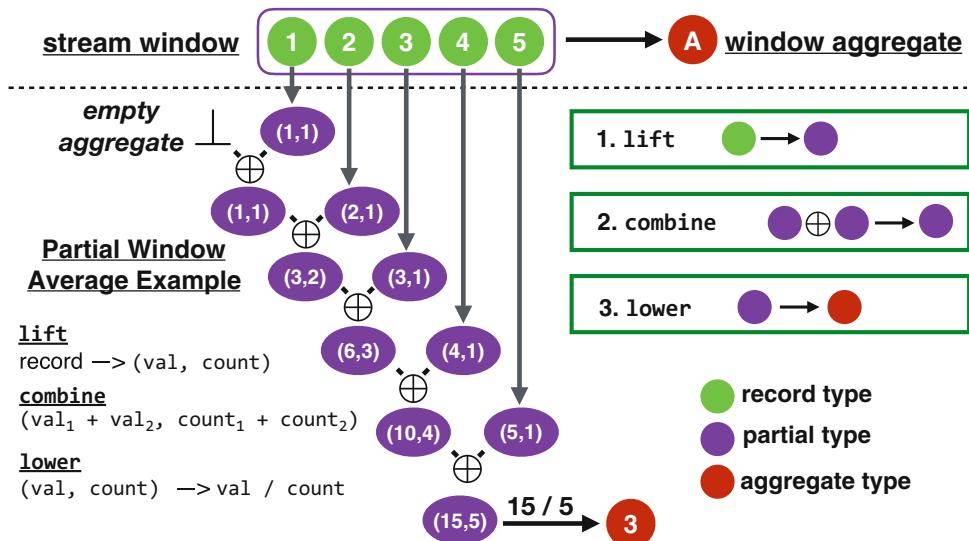
$\text{lift} : T \rightarrow A$ maps an element of a window to a partial aggregate of type A .

$\text{combine } \oplus : A \times A \rightarrow A$ combines two partial aggregates into a new partial aggregate (equivalent to a `reduce` function).

$\text{lower} : A \rightarrow T'$ maps a partial aggregate into an element in the type T' of output values.

The main and only requirement for partial aggregation is to have an *associative* `combine` function so that aggregation can be used to evaluate a full window aggregate in discrete steps (Arasu and Widom, 2004; Krishnamurthy et al., 2006; Tangwongsan et al., 2015).

An example of partial aggregation of a window is depicted in Fig. 3. The goal in this example is to partially compute the average value out of a set of records with values 1 to 5. The invariant is that only one partial aggregate is kept in memory (initially an empty aggregate). That aggregate is incrementally updated by each record that arrives in a window. To compute an average, two values have to be maintained in the aggregate type: a *sum* and a *count*. `lift` function, each record is first mapped into an aggregate type of its value



Stream Window Aggregation Semantics and Optimization, Fig. 3 Partial aggregation example for window average

and a count of 1. The `combine` function updates the partial aggregate with the new sum and count until all elements of a window have arrived. Then finally, the `lower` function transforms the aggregate into the window average, in this case this is 3.

Window Slicing

The amount of overlapping across sliding windows introduces additional space and computational complexity that partial aggregation itself cannot solve.

In the case of windows with predefined periodic characteristics such as a time or count slide, a family of optimization techniques is used to further decompose windows into nonoverlapping partial aggregates which can be shared and combined to calculate full window aggregates. This technique is typically named “slicing” or “bucketing.” The two most popular slicing techniques that have also been deployed in production stream processing systems in the past are *panes* (Li et al., 2005a) and *pairs* (Krishnamurthy et al., 2006).

Panes

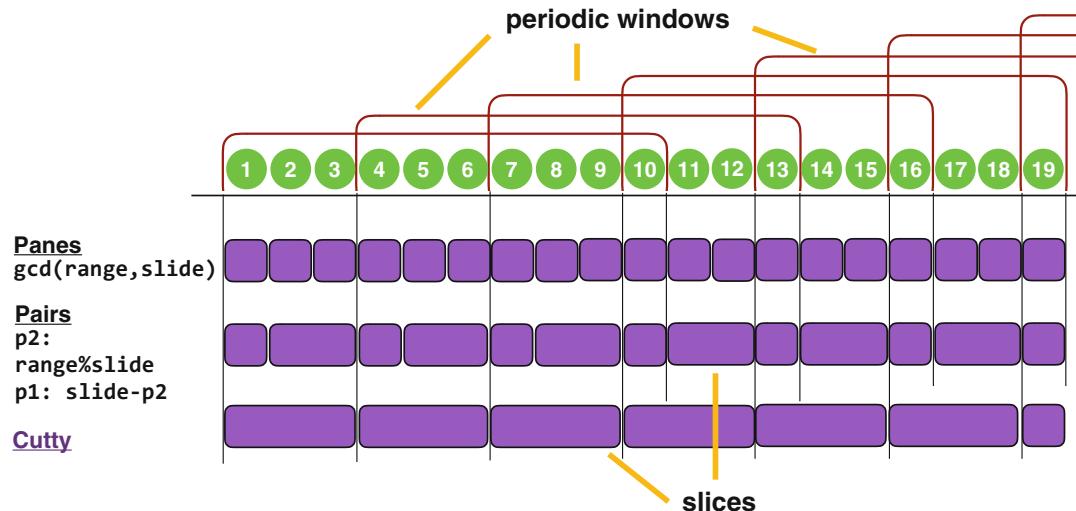
The main idea behind *Panes* is that if we have a periodic window query with a fixed slide and a

range, it is trivial to break down the aggregation process into partials with a constant size, equal to the greatest common denominator of the respective *range* and *slide*. For example, if we have a sliding window with a range of 9 min and a slide of 6 min, the stream would be sliced and pre-aggregated into buckets, each of which corresponds to 3 min of the ingested stream (greatest common denominator of 6 and 9).

Panes have been criticized (Krishnamurthy et al., 2006) for their lack of general applicability, yielding unbalanced performance that depends highly on the combination of range and slide. For example, a window of range 10 s and a slide of 3 s would break down to slices of a single second, no longer exploiting the amount of nonoverlapping segments in a stream (as depicted in Fig. 4).

Pairs

The *pairs* technique (Krishnamurthy et al., 2006) splits a stream into two alternating slices: $p_2 = \text{range} \bmod \text{slide}$ and $p_1 = \text{slide} - p_2$. This technique utilizes better nonoverlapping segments in a stream and seems to work well with most combinations of range and slide. Intuitively, pairs break slicing only when a stream window starts or ends. This is visualized in Fig. 4 where



Stream Window Aggregation Semantics and Optimization, Fig. 4 Example of different slicing techniques

it results into a lower number of slices for the same window compared to using *panes*. Contrary to *panes*, *pairs* has also been proposed for multi-query aggregation sharing where a large sequence of shared slices is decided at compilation time across shared sliding window aggregation queries in the same operator.

Slicing Limitations

While slicing offers the best known space and computational performance in sliding window aggregation, its applications are limited to periodic window queries since this is the only type of windows for which their beginning and end are predefined. The *Cutty* technique (Carbone et al., 2016) which is further analyzed in section “[Cutty: A Hybrid Approach](#)” avoids this strong assumption by letting user-defined functions signify during runtime when windows start or end. In the same work slicing is also combined with general pre-aggregation techniques (analyzed in section “[General Pre-aggregation](#)”) in order to combine the strongest characteristics of both domains of window optimization.

General Pre-aggregation

The main incentive of general pre-aggregation techniques is to be able to allow for arbitrary

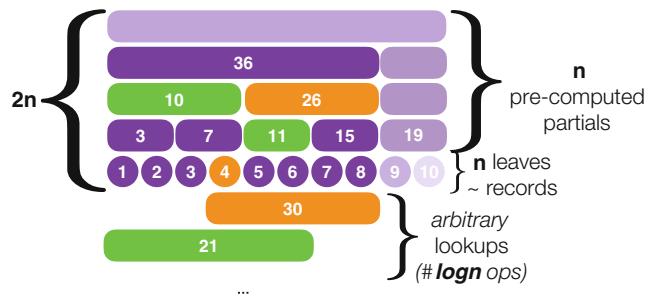
segment lookups in a stream (e.g., from external user queries) as depicted in Fig. 5.

The earliest work on general pre-aggregation was presented in *B-int* (Arasu and Widom, 2004) which precomputes eagerly higher-order partials on different segments of a stream. The application of *B-int* was meant to be fast aggregate retrieval for ad hoc stream queries (i.e., using CQL); however, the applicability of general pre-aggregation makes such techniques convenient for aggregating continuous sliding windows without a known range or slide or other periodicity assumptions. The Reactive Aggregator by IBM Research (Tangwongsan et al., 2015) exploits the properties of *B-Int* and introduces FlatFat: a fixed size circular heap binary tree of higher-order partials that “slides” together with the records of the stream.

General Pre-aggregation Limitations

General pre-aggregation offers fast retrievals of arbitrary windows on a stream at the cost of additional space and incremental update computation requirements. That is due to the fact that every time a new aggregate is added to the binary tree a sum of $\log(N)$ additional partial aggregations need to be employed in order to update all higher-order aggregates of the tree (given N : the number of active records/leaves in the tree). In summary,

Stream Window Aggregation Semantics and Optimization, Fig. 5
Example of a general pre-aggregation tree



the runtime costs of employing eager aggregation, which are also visualized in Fig. 5 are the following:

space: $2N$ partials need to always be kept on heap to hold the full aggregation binary tree.

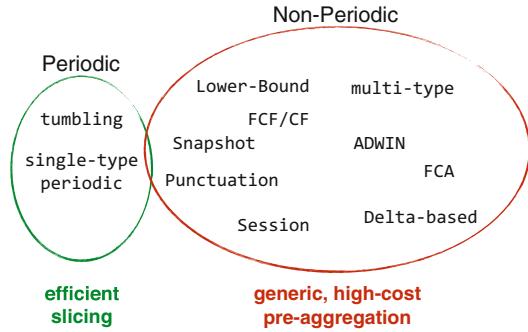
update/lookup: both update and full window lookup have $O(\log N)$ computational complexity. In the case of updates, the complexity is fixed ($\log N$) against slicing which typically involves a single aggregation per record.

All these costs pose an interesting trade-off when eager aggregation is employed per record in a data stream, which often results in more operations than a naive execution of each redundant window aggregation separately. As a result, it is likely that if general pre-aggregation is de facto applied, its runtime cost would never be amortized across a full run of a continuous application.

Nevertheless, the power of general pre-aggregation lies at the observation that it can be generally applied to any type of windows, thus, covering a large space of nonperiodic window types used within research and industrial applications, as depicted in Fig. 6.

Cutty: A Hybrid Approach

Slicing and general pre-aggregation are orthogonal techniques that can be potentially combined to support a wider variety of stream windows for aggregation. The *Cutty* aggregator (Carbone et al., 2016) employs such a hybrid approach that can lead to efficient aggregation of a broader number of window types than simply periodic. The main observation behind its design is that

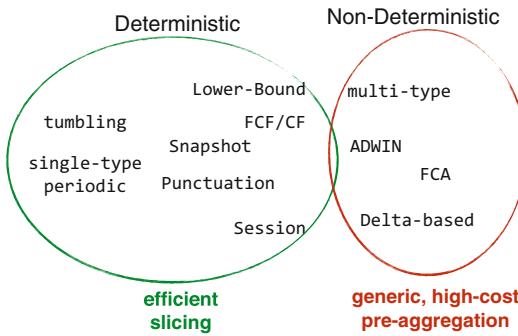


Stream Window Aggregation Semantics and Optimization, Fig. 6 Applicability of slicing and general pre-aggregation

there is an implicit class of windows (a superclass of periodic ones), termed *deterministic* which can be obtained by using the right core primitives in the programming model. Deterministic windows can be used to enable efficient shared aggregation without limiting window expressivity. Figure 7 shows the expressive power of deterministic windows, being able to provide optimal pre-aggregation to more than the limited periodic windows.

Deterministic Windows

The concept of deterministic windows stems from the observation that all it takes to achieve optimal slicing is not the *a priori* knowledge of the periodicity of windows (if any) but the *runtime* knowledge of where a new window starts. While partial aggregation is employed, as described in section “[Partial Window Aggregation](#)”, a single partial result can be kept in active memory until we receive a record that marks the beginning of a new window. Conceptually, a new window start means that



Stream Window Aggregation Semantics and Optimization, Fig. 7 Visualizing the expressive power of Deterministic Windows for Efficient Aggregation

we will later need a partial aggregate (or slice) starting at that point to evaluate the window that started there.

Cutty proposes user-defined windowing through the use of a *discretization function* f_{disc} , defined as follows:

$$f_{\text{disc}} : T \rightarrow \langle W_{\text{begin}} : \mathbb{N}, W_{\text{end}} : \mathbb{N} \rangle$$

where for each record $r \in T$: (i) W_{begin} is the number of windows beginning with r and (ii) W_{end} the number of windows ending with r .

Overview of Cutty

Optimal slicing with deterministic windows minimizes but does not eliminate redundancy. As an example, consider the slices produced by *Cutty* during the example execution of Fig. 4. A detailed observation of the slices used per window reveals a level of redundancy that cannot be handled by slicing. For example, slices $s[4, 6]$ and $s[7, 9]$ would have to be combined together twice: once for computing $f_a(s[1, 10])$ itself and once for computing $f_a(s[4, 13])$. Instead, if somehow the evaluation of $f_a(s[4, 9])$ was stored, it would not be necessary to repeat that aggregation.

Cutty utilizes general pre-aggregation (FlatFat) in order to further reduce the cost of full window aggregate evaluation and compute higher-order combinations of aggregates only once. This idea gives a new purpose to general pre-aggregation techniques and grants a low memory footprint since the space complexity of

the aggregation tree is bounded by the number of active slices (which is equivalent to the minimum number of nonoverlapping segments in a stream). A full example run of *Cutty* is visualized in Fig. 8, showing both slicing of deterministic windows and general pre-aggregation and evaluation on stored partials. In the same example, notice that the partial $P(s[6, 7])$ is computed once and reused for both $f_a(s[1, 5])$ and $f_a(s[1, 6])$.

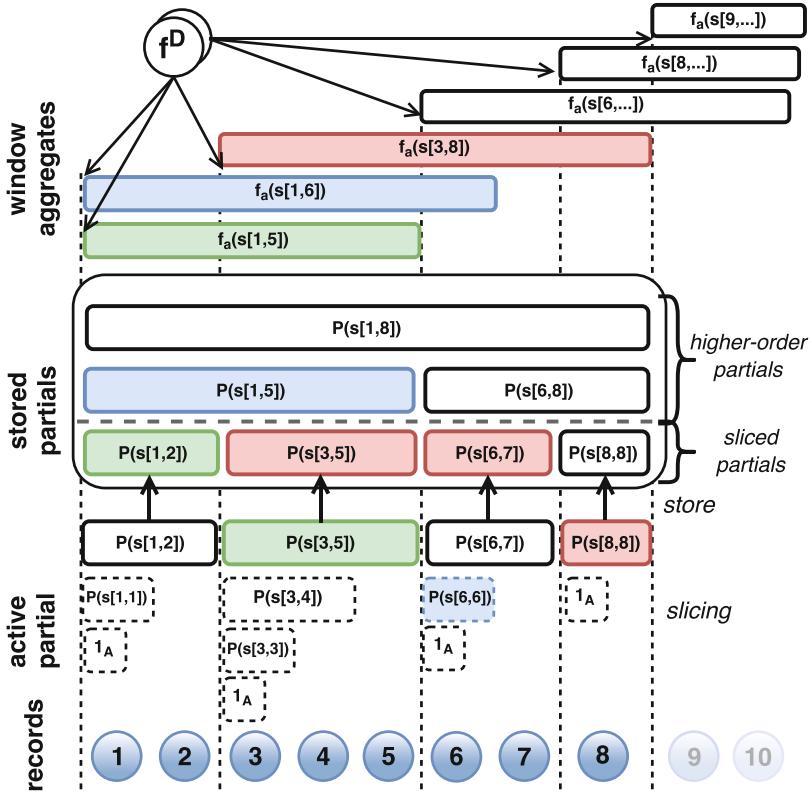
For nondeterministic windows, *Cutty* falls back to general pre-aggregation (simply using FlatFat) since slicing cannot be applied. Nevertheless, the generality and flexible (runtime specific) nature of this aggregation technique also enables the prospect of using it for applying operator sharing (Hirzel et al., 2014) on data streams. A full complexity and performance analysis and comparison are provided in the original *Cutty* paper (Carbone et al., 2016).

Further Works

Window aggregation is an interesting research topic, and there are many relevant proposed ideas to the ones presented here. For example, Li et al. (2005b, 2008a) classified window types by their evaluation context requirements, leaving the characterization of each class as an open research question. Performing certain types of aggregates in constant-time was recently proposed (Tangwongsan et al., 2017). Deterministic functions in *Cutty* subsume all forward-context-free windows (no future records are required to know when a window starts), while nondeterministic discretization functions are forward-context-aware. Heuristic-based plan optimizers have also been proposed (e.g., *TriWeave* Guirgis et al. 2012) to fine-tune the execution of periodic time queries dynamically using runtime metrics (i.e., input rate and shared aggregate rate).

Future Directions

Windowing semantics are becoming increasingly more complex and sophisticated as data stream



Stream Window Aggregation Semantics and Optimization, Fig. 8 An overview example of *Cutty*

processing is widely adopted. Aggregation techniques will have to follow the trends in windowing semantics and adapt to more dynamic, data-centric window types. One of the most prominent future directions in stream windowing is its standardization and encapsulation in stream SQL standards that are undergoing in open-source communities (e.g., the Calcite project and Google Dataflow Akidau et al. 2015). However, no significant efforts have been made to apply relational optimizations in stream windowing. Another future direction is to extend slicing capabilities beyond deterministic windows (if possible) and cover cases of fully data-driven windows without FIFO guarantees such as ADWIN (Bifet and Gavalda, 2007). Finally, general pre-aggregation data structures have to employ the notion of out-of-orderness (Traub et al., 2018). Currently, with existing out-of-the-box solution such as FlatFat, it is not possible to retract already evaluated window

aggregates, thus, making it impossible to use for systems like Beam and Flink with out-of-order logic.

Conclusions

Windows over streaming data continue to be the most central abstraction in data stream processing. Aggregation techniques aim to reduce the computational redundancy to the maximum extent possible for sliding windows. Most often, approaches to efficient aggregation are entangled with actual windowing semantics, such as assuming periodic queries to provide efficient pre-aggregation. Slicing techniques provide low memory footprint and generally good performance at the cost of limited applicability, while general pre-aggregation techniques can be employed for any window lookup at the cost of high computational and

memory footprint. Recent approaches aim for a hybrid solution by generalizing slicing further while combining data structures from general pre-aggregation. Sliding window aggregation remains a challenging topic today, and new challenges will arise with the adoption of richer and more complex windowing semantics and out-of-order streams.

Cross-References

► Sliding-Window Aggregation Algorithms

References

- Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In: VLDB
- Arasu A, Widom J (2004) Resource sharing in continuous sliding-window aggregates. In: VLDB
- Arasu A, Babcock B, Babu S, Cieslewicz J, Datar M, Ito K, Motwani R, Srivastava U, Widom J (2016) Stream:the Stanford data stream management system. In: Data stream management. Springer, Berlin/Heidelberg, pp 317–336
- Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. In: VLDBJ
- Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: SDM. SIAM
- Botan I, Derakhshan R, Dindar N, Haas L, Miller RJ, Tatbul N (2010) Secret: a model for analysis of the execution semantics of stream processing systems. In: VLDB
- Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink: stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Commun Data Eng 36(4):28–38
- Carbone P, Traub J, Katsifodimos A, Haridi S, Markl V (2016) Cutty: aggregate sharing for user-defined windows. In: Proceedings of the 25th ACM international conference on information and knowledge management. ACM
- Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in Apache Flink®: consistent stateful distributed stream processing. Proc VLDB Endow 10(12):1718–1729
- Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein JM, Hong W, Krishnamurthy S, Madden SR, Reiss F, Shah MA (2003) TelegraphCQ: continuous dataflow processing. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data. ACM, pp 668–668
- Guirguis S, Sharaf MA, Chrysanthis PK, Labrinidis A (2012) Three-level processing of multiple aggregate continuous queries. In: IEEE ICDE
- Hirzel M, Andrade H, Gedik B, Kumar V, Losa G, Nasgaard M, Soule R, Wu K (2009) SPL stream processing language specification. New York: IBMResearchDivisionTJ WatsonResearchCenter, IBM ResearchReport: RC24897 (W0911–044)
- Hirzel M, Soulé R, Schneider S, Gedik B, Grimm R (2014) A catalog of stream processing optimizations. ACM Comput Surv (CSUR) 46(4):46
- Krishnamurthy S, Wu C, Franklin M (2006) On-the-fly sharing for streamed aggregation. In: AMC SIGMOD
- Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005a) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. ACM SIGMOD Rec 34:39–44
- Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005b) Semantics and evaluation techniques for window aggregates in data streams. In: ACM SIGMOD
- Li J, Tufte K, Maier D, Papadimos V (2008a) Adaptwid: an adaptive, memory-efficient window aggregation implementation. IEEE Internet Comput 12:22–29
- Li J, Tufte K, Shkapenyuk V, Papadimos V, Johnson T, Maier D (2008b) Out-of-order processing: a new architecture for high-performance stream systems. Proc VLDB Endow 1(1):274–288
- Tangwongsan K, Hirzel M, Schneider S, Wu KL (2015) General incremental sliding-window aggregation. In: VLDB
- Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Proceedings of the 11th ACM international conference on distributed and event-based systems. ACM, pp 66–77
- Traub J, Grulich P, Rodriguez Cuellar A, Bress S, Katsifodimos A, Rable T, Markl V (2018) Scotty: efficient window aggregation for out-of-order stream processing. In: 2012 IEEE 34th international conference on data Engineering (ICDE). IEEE

Stream-Based Microservices

► Streaming Microservices

Streaming Architecture

► Streaming Microservices

Streaming Big Spatial Data

Ahmed R. Mahmood and Walid G. Aref
Purdue University, West Lafayette, IN, USA

Synonyms

Distributed spatial data streaming; Real-time big spatial data processing

Definitions

Recently, several big data stream management systems (DSMS, for short) have been developed to provide an infrastructure to process streamed big data. Big spatial DSMSs constitute a special class of big DSMSs that are optimized to process large amounts of spatial data streams. The main idea behind most big spatial DSMSs is to leverage the spatial properties of the incoming data stream to fairly distribute the workload across multiple distributed processes. When processing big spatial data streams, it is important to maintain high throughput and low latency.

Overview

Spatial data is ubiquitous. It is continuously being generated at a large scale. This is due to the popularity of GPS-enabled devices, e.g., smartphones, smart-watches, personal activity trackers, and GPS-navigation devices. Efficient processing of this streamed big spatial data requires higher computational resources than the resources that exist in a centralized system.

Big spatial DSMSs can be classified into the following two main categories: (1) *dedicated big spatial DSMSs* and (2) *spatial extensions to general-purpose big DSMSs*.

- *Dedicated big spatial DSMSs*. These systems are the first step toward scalable spatial data

stream processing. These distributed big spatial DSMSs are built from scratch with the sole purpose of processing streamed big spatial data. These systems often lack important features that exist in general-purpose big DSMSs, e.g., fault tolerance, and the ability to perform non-spatial data processing. Also, designers of these systems implement the communication protocols between the distributed processes. These communication protocols often require serialization, synchronization, and encryption. These requirements are orthogonal to big spatial data stream processing and require a substantial development overhead.

- *Spatial extensions to general-purpose big DSMSs*. In this widely adopted approach, general-purpose big DSMSs, e.g., Storm (Toshniwal et al., 2014), Spark-Streaming (Zaharia et al., 2012), and Yahoo S4 (Neumeyer et al., 2010), are extended with spatial distribution approaches to optimize the processing of spatial queries over spatial data streams. When extending an existing big DSMS with spatial capabilities, one inherits the features of the underlying general-purpose big DSMS, e.g., high scalability, fault-tolerance, and robust communication protocols.

The streamed spatial data is often associated with textual data, e.g., as in the case of tweets. A tweet contains textual data as well as the user's location where the tweet is issued. Big spatio-textual DSMSs are designed to account for the spatial and the associated textual attributes to optimize the processing of spatio-textual queries over spatio-textual data streams.

Key Scientific Findings

First, we describe the main query types over spatial data streams. Then, we give a historical background on general-purpose big DSMSs and centralized spatial DSMSs. Then, we focus on the categories of big spatial DSMSs.

Query Types over Spatial Data Streams

There is a wide range of spatial predicates that can be used when querying spatial data streams. The two main classes of spatial predicates are spatial selects and spatial joins.

1. *Spatial Select*: In a spatial select, given a spatial data stream and a spatial predicate, the spatial select produces as output the data tuples from the spatial data stream that satisfy the spatial predicate. Examples of a spatial select are the spatial range select and the top-k nearest neighbor select.

The range select predicate operates on a single spatial data stream and has a specific spatial range predicate. This select predicate identifies the streamed data tuples (i.e., the streamed spatial objects, e.g., cars or shops) that satisfy the spatial range predicate, i.e., the spatial objects that are located inside the specified spatial range.

The k-nearest-neighbor (kNN) select predicate operates on a single spatial data stream and has a focal point and a parameter k . The kNN spatial select identifies the k -nearest data objects to the focal point specified. The distance function used in measuring the distance from the focal point to the spatial data objects can vary. It can be the Euclidean distance or the distance over an underlying road network.

2. *Spatial Join*: In a spatial join, given two spatial data streams and a spatial join predicate, the spatial join produces as output data tuple pairs, one from each spatial data stream, such that each pair of tuples satisfies the spatial join condition. The spatial join has many flavors based on the type of spatial join predicate that is being used. The following are two popular spatial join operation.

The spatial range join: The spatial predicates can be of various forms. For example, one spatial predicate can be a distance threshold, say d . In this case, the spatial join will produce as output the pairs of objects, say (x_i, y_j) , where Object x_i from the first data source is within distance d from Object y_j from the second data source. This type of operation is referred to as a spatial range join operation.

The spatial k-nearest neighbor join (kNN-join): Depending on the spatial predicate involved, other types of spatial joins may be formed. For example, the spatial kNN-join operation produces as output the pairs of objects, say (x_i, y_j) , where Object y_j from the second data source is among the k -closest objects to Object x_i from the first data source. In addition to the two input spatial data streams, the number k is also specified as an input parameter to the spatial kNN-join operation. Any distance function can be used. Examples are the Euclidean or Manhattan distances or the distance over an underlying road network graph.

A single spatial data source may be used in the spatial join in contrast to two data sources. In this case, it is termed a spatial self-join. For example, in a spatial range self-join, pairs of spatial objects from the same source that are within distance d from each other are reported as output. For all the above operations, various distance functions can be used, e.g., the Euclidean, Manhattan, and road network distance metrics.

Continuous vs. Snapshot Spatial Queries

Spatial queries over data streams can either be evaluated as snapshot or continuous queries. A snapshot query operates on the current state of the spatial stream and produces a single resultset. In contrast, a continuous spatial query operates continuously for a relatively long duration, where the resultset of the continuous query is continuously updated to reflect the incoming spatial data from the streamed spatial data source.

To evaluate a snapshot spatial query, the state of the streamed spatial objects is maintained (usually using a spatial index). For example, consider a stream of location updates of moving objects. The state of the stream could be the current locations of the moving objects or a limited history of the trajectories of the moving objects. When a snapshot query arrives, it is evaluated once over the current state of spatial data stream and produces as output the spatial data objects that qualify the query. Finally, the snapshot query is completely evaluated, and it quits.

In contrast, when a continuous query arrives, it is stored into the system and is progressively evaluated against the incoming spatial data objects. The DSMS maintains a state for each of the continuous queries. This state may include some intermediate summary data as well as the current resultset of the query. For example, in the case of continuous query that contains a kNN select predicate, the DSMS maintains the current set of k spatial objects that are closest to the focal point. Because the query is continuous, as the objects move, the current set of k -nearest objects to the focal point continues to get updated.

Continuous spatial queries are often indexed (using a spatial index) to speed up the maintenance of the arriving spatial objects, e.g., the continuous updates of the locations of the spatial objects as they move in space. Having a spatial index facilitates the progressive evaluation of the continuous spatial query.

Static vs. Dynamic Spatial Queries

Continuous spatial queries over spatial data streams can either be static or dynamic. A static continuous spatial query does not change its input over time. For example, the spatial location of a kNN select predicate remains constant over time for the entire lifetime of the continuous spatial query. In contrast, a dynamic continuous spatial query may change its input parameters over time. For example, consider the following query: Find the three-closest gas stations to my current location while I am driving on the highway. This continuous query is composed of a kNN select predicate, where the focal point of the kNN select corresponds to the location of a moving object. Changes to the location of this focal point as the object moves may result in updating the query's resultset even without having any updates from the spatial data source.

Historical Background

General-Purpose Big DSMSs

Several general-purpose big stream management systems have been extended to support big spatial data stream processing. These systems provide an

infrastructure for the scalable processing of data streams. Generally, the main advantages of these systems are (1) hiding the complexity of transferring data between distributed processes, (2) high throughput, and (3) reliability and fault tolerance. Two main data processing models are currently being adopted by general-purpose big DSMSs, namely, *tuple-based* and *micro-batching*.

In the *tuple-based* streaming model, every tuple in the stream can be processed individually. Yahoo S4 (Neumeyer et al., 2010) and Apache Storm (Toshniwal et al., 2014) are examples of tuple-based big data streaming systems. The main advantage of the tuple-based model is the low processing latency of streamed tuples. The main limitation of this model is that a single tuple may be processed more than once in certain failure situations.

In the *micro-batching* streaming model, tuples are not processed individually but are rather buffered for a time duration (up to a few seconds) to create a micro-batch. M3 (Aly et al., 2012) is a main-memory map-reduce-based system that modifies (Apache Hadoop, 2017) in the following way. It removes the HDFS layer and replaces it by memory buffers that are maintained in a distributed layer of data nodes. Data continues to accumulate until the memory buffers are filled or until some maximum delay threshold is reached. Then, data is batched into the map-reduce layer. In M3, the disk layer between the mappers and the reducers is eliminated and is replaced by another layer of distributed memory buffers. Data is communicated between the mappers' and the reducers' distributed memory buffers using networking sockets. SparkStreaming (Zaharia et al., 2012) is another example of a micro-batching-based DSMS. SparkStreaming extends Spark, a general-purpose main-memory big data system. SparkStreaming inherits the resilient distributed datasets (RDD) and lineage from Spark to ensure fault tolerance in stream processing. RDD is an immutable main-memory data structure that represents a collection of objects. Processing in Spark and SparkStreaming is performed by applying transformations to RDDs. Lineage is a graph that represents transformations to RDDs. In failure scenarios, the lineage graph is consulted

to reconstruct the failed RDDs. SparkStreaming builds RDDs over micro-batches of streams every specific time duration (up to a few seconds). The main advantages of this model are high throughput and that every tuple gets processed exactly once even under failure scenarios. However, the main limitation of micro-batching is the high latency incurred while buffering streamed data to build a micro-batch.

Centralized Spatial DSMSs

Several centralized spatial data stream management systems have been developed to answer spatial queries over spatial streams. Examples of these systems include PLACE (Mokbel et al., 2004b), SINA (Mokbel et al., 2004a), SEA-CNN (Xiong et al., 2005), SOLE (Mokbel and Aref, 2008), and Gpac (Mokbel and Aref, 2005). These systems are not designed for scalability, and their performance is constrained by the resources of the single machine they run on.

Big Spatial DSMSs

We describe the main types of big spatial DSMSs below.

Dedicated Big Spatial DSMSs

Special-purpose big spatial DSMSs are distributed systems that can process only spatial data streams. These systems often address specific types of queries over spatial data streams. They focus on the scalability of spatial query processing and do not address other requirements of distributed big data processing, e.g., the efficient scheduling of distributed processes across the cluster of machines, reliability, fault tolerance, error recovery, or fair workload distribution across the processes. These systems are considered the first step toward big spatial DSMSs. Examples of these dedicated and spatial-purpose distributed big spatial DSMSs include PLACE* (Xiong et al., 2007), MobiPLACE (Zakhary et al., 2013), and MobiEyes (Gedik and Liu, 2006).

PLACE* (Xiong et al., 2007) use multiple servers to evaluate continuous moving range queries (dynamic queries) over moving objects in the Euclidean space. This system employs an incremental update model that updates query results only when there is a change to the resultset of the queries. MobiPLACE (Zakhary et al., 2013) applies the processing model of PLACE* over road networks.

MobiEyes (Gedik and Liu, 2006) use a centralized server and multiple small smart-mobile devices. To reduce the workload at the centralized server, MobiEyes offloads and distributes the processing of the continuous moving spatial range queries to the mobile devices.

Spatial Extensions to Big DSMSs

The most adopted approach in big spatial data streaming is to extend a general-purpose big data stream management system. This allows spatial extensions to inherit the scalability and reliability features of the underlying streaming platform. Spatial extensions to general-purpose big data stream management systems differ in the streaming model supported, i.e., tuple based or micro-batching based. These systems also differ in the type of spatial queries supported, the type of the spatial indexes used, and whether the spatial extension is adaptive and sensitive to the change in distribution of the data and the query workload of the spatial data stream or not.

Micro-Batching-Based Systems

Grid-based indexing on top of SparkStreaming (Choi et al., 2015; Lee and Song, 2015). These systems answer continuous range queries over spatial data streams on top of SparkStreaming. They use grid-based partitioning to distribute the streamed workload across the worker processes of the underlying cluster.

Cruncher (Abdelhamid et al., 2016). This system is an adaptive extension to SparkStreaming that addresses kNN and range queries over a spatial data stream. Cruncher keeps statistics about the spatial data and query workload to build a kd-tree-based (Ooi et al. 1987) spatial partitioning over the spatial data and queries. This partitioning is applied to every micro-batch to create

sub-micro-batches. These sub-micro-batches are fairly distributed across the worker processes of the underlying spark cluster. Cruncher also uses the spatial properties of continuous range queries to share the execution among the spatial queries that have overlapping ranges.

Tuple-Based Systems

Algorithms for spatial queries on top of Storm (Zhang et al., 2016). This approach develops algorithms to answer continuous spatial queries on top of Storm. The queries supported are the spatial range, kNN, and the spatial join. The current locations of the moving objects are maintained in separate Storm processes. Each Storm process is responsible for a specific set of moving objects. Within every Storm process, say P , a spatial index is used to maintain the current locations of the moving objects handled by P . Spatial queries are replicated to all Storm processes to produce partial results. Partial results of these queries are collected from the distributed Storm processes to be aggregated at dedicated aggregation processes. To process a spatial join operation, a different partitioning algorithm is used. Instead of partitioning the moving objects based on their identifiers, moving objects are partitioned spatially using grid-based partitioning. The range of a spatial join operation may span multiple Storm processes. Moving objects that overlap the spatial range of the spatial join operation are paired together to produce the final answer.

Continuous kNN-join processing on top of Storm (Song, 2016) This approach develops a parallel algorithm to evaluate continuous kNN-join operations on top of spatial streams. The algorithm uses data partitioning and replication of distributed processes to group in the same processes all the possible pairs that can be joined together. This algorithm is realized on top of Storm.

DSI: kNN processing on top of S4 (Yu et al., 2015). DSI is a distributed spatial partitioning mechanism that is realized on top of Yahoo S4 (Neumeyer et al., 2010). DSI processes snapshot kNN operations over the current locations of moving objects. DSI partitions the

space into horizontal and vertical strips. Every horizontal and vertical strip is assigned to a distributed process in S4. To process a kNN operation, partial kNN results are calculated from the strips. Then, a global kNN result is aggregated from all the partial results. DSI is adaptive and attempts to ensure fair workload distribution by updating the boundaries of the horizontal and vertical strips based on the changes in the workload.

kNN processing on top of IBM System S (Wu et al., 2012). In this approach, the continuous kNN operation is handled as a stateful operator, i.e., it maintains the current set of the k-nearest objects closest to the focal point of the continuous kNN operation. A distributed version of the kNN operation is realized using IBM System S. IBM System S is a proprietary general-purpose big data stream management system. To evaluate this kNN operation, location updates of moving objects are partitioned using a hash function to multiple processes. Then, an aggregation step is used to find the global kNN resultset.

Big Spatio-textual Data Streaming

Spatial data streams are often associated with a textual attribute. For example, tweets have both spatial and textual attributes. Several big spatial data stream management systems have been developed to process streamed spatio-textual data. The main feature in these systems is that they utilize the textual attribute of the spatio-textual data to optimize the processing of spatio-textual queries.

The two main spatio-textual queries being supported by big spatial data stream management systems are (1) *the continuous spatio-textual filter query* and (2) *the continuous top-k spatio-textual query*.

The continuous spatio-textual filter query has both an input spatial range and a set of input keywords. In this query, it is required to identify the streamed spatio-textual objects located inside the spatial range of the query and contain all of the keywords of the query. Tornado (Mahmood et al., 2015, 2017) and PS²Stream (Chen et al., 2017) are two big spatial data stream management sys-

tems that address these queries. Both systems extend Storm. The main idea behind these systems is to evenly distribute the spatio-textual data and queries over distributed Storm processes. This is performed using a global distribution layer that contains a spatio-textual distribution index. Within every worker process, continuous spatio-textual filter queries are indexed using an internal spatio-textual index.

The continuous top-k spatio-textual query. This query has an input focal point and an associated set of input keywords. This query maintains a time-sliding window over a spatio-textual data stream. In this query, it is required to continuously identify the most relevant spatio-textual data objects within the time-sliding window. Relevance between the objects and the query is calculated using a function of the spatial distance and the textual similarity between the spatio-textual data objects and the queries. DSkye (Wang et al., 2017) is an extension to Storm. DSkye uses a global spatio-textual distribution mechanism to distribute stream spatio-textual queries to worker processes. In worker processes, a local index stores the spatio-textual objects for a specific time-sliding window. For every incoming query, the distributed local indexes are searched to identify an initial list of the top-k relevant data objects. Then, the list of the top-k relevant data objects keeps being updated according to the incoming spatio-textual streamed data objects.

Examples of Applications

Many applications require the processing of this streamed big spatial data in real time. Example applications include finding travel companions; ride-sharing, e.g., Uber; real-time navigation and map services, e.g., Google Maps; and real-time traffic analysis. Nowadays, algorithms adopted in online advertisement targeting consider the locations of users to identify relevant local advertisements. Real-time analysis of microblogs, e.g., identify trending Twitter hashtags within a specific location, is one important

application of processing streamed spatio-textual data.

Future Directions for Research

One important research direction is online big spatial data analytics. The challenge here is to support a richer set of complex spatial predicates that also involves aggregations. One challenge of supporting these spatial predicates is to devise effective cost estimation techniques that can capture the varying overhead of these operations. These cost estimation techniques are crucial to maintaining workload balance over the distributed processes of the underlying spatial DSMSs.

Also, existing big spatial DSMSs do not efficiently support the temporal dimension. One example scenario of spatio-temporal and textual processing is the continuous aggregation of streamed spatial data based on their textual attributes over a time-sliding window. This type of data management requires novel distributed spatial-temporal and textual distribution, indexing, and query processing algorithms that account for the peculiarities of the temporal dimension.

Cross-References

- ▶ [Apache Samza](#)
- ▶ [Apache Spark](#)
- ▶ [Spatio-textual data](#)

S

References

- Abdelhamid AS, Tang M, Aly AM, Mahmood AR, Qadah T, Aref WG, Basalamah S (2016) Cruncher: distributed in-memory processing for location-based services. In: IEEE 32nd international conference on data engineering (ICDE). IEEE, pp 1406–1409
- Apatche Hadoop (2017) Apatche Hadoop. <http://hadoop.apache.org/>
- Aly AM, Sallam A, Gnanasekaran BM, Nguyen-Dinh LV, Aref WG, Ouzzani M, Ghafoor A (2012) M3: stream processing on main-memory mapreduce. In: ICDE, pp 1253–1256
- Chen Z, Cong G, Zhang Z, Fuz TZ, Chen L (2017) Distributed publish/subscribe query processing on the

- spatio-textual data stream. In: IEEE 33rd international conference on data engineering (ICDE). IEEE, pp 1095–1106
- Choi D, Song S, Kim B, Bae I (2015) Processing moving objects and traffic events based on spark streaming. In: 8th international conference on disaster recovery and business continuity (DRBC). IEEE, pp 4–7
- Gedik B, Liu L (2006) Mobieyes: a distributed location monitoring service using moving location queries. *IEEE Trans Mobile Comput* 5(10):1384–1402
- Lee Y, Song S (2015) Distributed indexing methods for moving objects based on spark stream. *Int J Contents* 11(1):69–72
- Mahmood AR, Aly AM, Qadah T, Rezig EK, Daghstani A, Madkour A, Abdelhamid AS, Hassan MS, Aref WG, Basalamah S (2015) Tornado: a distributed spatio-textual stream processing system. *PVLDB* 8(12): 2020–2023
- Mahmood AR, Daghstani A, Aly AM, Aref WG, Tang M, Basalamah S, Prabhakar S (2017) Adaptive processing of spatial-keyword data over a distributed streaming cluster. arXiv preprint, arXiv:170902533
- Mokbel MF, Aref WG (2005) Gpac: generic and progressive processing of mobile queries over mobile data. In: Proceedings of the 6th international conference on mobile data management. ACM, pp 155–163
- Mokbel MF, Aref WG (2008) Sole: scalable on-line execution of continuous queries on spatio-temporal data streams. *VLDB J* 17(5):971–995
- Mokbel MF, Xiong X, Aref WG (2004a) Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data. ACM, pp 623–634
- Mokbel MF, Xiong X, Aref WG, Hambrusch SE, Prabhakar S, Hammad MA (2004b) Place: a query processor for handling real-time spatio-temporal data streams. In: Proceedings of the thirtieth international conference on very large data bases, VLDB endowment, vol 30, pp 1377–1380
- Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: IEEE international conference on data mining workshops (ICDMW). IEEE, pp 170–177
- Ooi BC, McDonell KJ, Sacks-Davis R (1987) Spatial kd-tree: an indexing mechanism for spatial databases. In: IEEE COMPSAC, sn. vol 87, p 85
- Song G (2016) Parallel and continuous join processing for data stream. PhD thesis, Université Paris-Saclay
- Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J et al (2014) Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 147–156
- Wang X, Zhang W, Zhang Y, Lin X, Huang Z (2017) Top-k spatial-keyword publish/subscribe over sliding window. *VLDB J* 26(3):301–326
- Wu S, Kumar V, Wu KL, Ooi BC (2012) Parallelizing stateful operators in a distributed stream processing system: how, should you and how much? In: Proceedings of the 6th ACM international conference on distributed event-based systems. ACM, pp 278–289
- Xiong X, Mokbel MF, Aref WG (2005) SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: Proceedings of the 21st international conference on data engineering, ICDE 2005. IEEE, pp 643–654
- Xiong X, Elmongui HG, Chai X, Aref WG (2007) Place: a distributed spatio-temporal data stream management system for moving objects. In: International conference on mobile data management. IEEE, pp 44–51
- Yu Z, Liu Y, Yu X, Pu KQ (2015) Scalable distributed processing of k nearest neighbor queries over moving objects. *IEEE Trans Knowl Data Eng* 27(5):1383–1396
- Zaharia M, Das T, Li H, Shenker S, Stoica I (2012) Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. *HotCloud* 12:10–10
- Zakhary V, Elmongui HG, Nagi MH (2013) Mobiplace*: a distributed framework for spatio-temporal data streams processing utilizing mobile clients processing power. In: International conference on mobile and ubiquitous systems: computing, networking, and services. Springer, pp 78–88
- Zhang F, Zheng Y, Xu D, Du Z, Wang Y, Liu R, Ye X (2016) Real-time spatial queries for moving objects using storm topology. *ISPRS Int J Geo-Inf* 5(10):178

Streaming Language

► [Stream Processing Languages and Abstractions](#)

Streaming Microservices

Ted Dunning and Ellen Friedman
MapR Technologies and Apache Software Foundation, Santa Clara, CA, USA

Synonyms

[Stream-based microservices](#); [Streaming architecture](#)

Definitions

Streaming microservices refers to a style of building large-scale software systems as a collection of independently deployable processes that commu-

nicate via persistent message streams rather than via remote procedure calls (RPC). A key distinction between streaming and RPC-based microservices is that streaming microservices provide a high degree of temporal decoupling between services. Not only does it not matter how a service does what it does, but it also matters much less exactly when it does it.

The use of streaming microservices is distinct from previous architectural trends such as Service Oriented Architecture (SOA) because the message streams used with streaming microservices are very simple, supporting little more than ordered delivery of messages organized into topics as opposed to the highly complex semantics supported by Enterprise Service Busses (ESB).

Historical Background

The term microservices as a style of systems architecture was introduced in 2011 at a conference in Venice and gained popularity around 2013–2014 (Fowler and Lewis, 2014) as people saw the success of this approach in large companies including Netflix, Amazon, and LinkedIn. Many of the core concepts underlying microservices for architecting large-scale systems go back at least a decade earlier, but the distinctive characteristics of microservice architectures were not well articulated until more recently.

The basic idea of microservices is to provide useful decompositions of large systems into independent services that could be started, stopped, and redeployed relatively independently. The key characteristic is that the implementation details of services are hidden. Virtues attributed to microservices include agility and flexibility in the development process by avoiding a monolithic approach with many development dependencies. Since services can be redeployed independently, the system is easier to upgrade progressively through evolutionary refinement of services one at a time. Teams working on different services can work to differing deadlines, thus avoiding many dependencies and schedule blocks.

In addition to the use of micro-services as a way to structure systems, they also define team organization. The result is that large systems devolve into a suite of loosely coupled small services, each built and run by a small, focused, cross-functional team. As originally viewed, microservices communicate via synchronous remote procedure calls (RPC), often implemented as REpresentation State Transfer (REST) (Fielding and Taylor, 2002) requests implemented over HTTP connections. More recently, asynchronous remote procedure calls have become much more common, especially in conjunction with node.js-based systems (Hughes-Croucher and Wilson, 2012).

The ability to deploy microservices independently makes development much easier than in a monolithic system or in a system that forces tight coupling between requestor and service. This is because individual services that can be upgraded independently also allow decoupling of development schedules. Additionally, using simple remote calls defined using systems like ProtoBufs, strict versioning of communication protocols is not necessary. Instead a more evolutionary soft-versioning approach can be taken that views all protocol changes as extensions and avoids complete redefinitions. Soft versioning of communication protocols provides additional decoupling of service development.

Decoupling of services also makes scaling easier, so explosive growth is a strong incentive to adopt microservice architectures. Selectively scaling individual services to match the scale of work that needs to be done by that service is much more efficient than scaling a monolithic service (Newman, 2015).

The final step to streaming microservices as opposed to RPC-based microservices is a much more recent development than the core concept of microservices. The fundamental addition that streaming brings is the idea that many services can actually be decoupled temporally as well as in terms of deployment. This sounds like a small change, but it substantially increases the scope of the microservice concept.

Foundations

Getting streaming microservices to work well requires a lexicon of high-level architectural patterns, a compatible underlying message transport, and methods for implementing the microservices themselves. This section describes each of these requirements in turn.

Emergence of Streaming Microservices

Widespread adoption of streaming microservices is a relatively recent trend in spite of the fact that the idea of moving data from one processing element to another is an old one (Johnston et al., 2004). What distinguishes modern streaming microservices from older dataflow concepts is that whereas dataflow was largely intended to build a system to execute a single computer program, a streaming microservices system is intended to support multiple high-level services that can be modified at any time. The idea of mutability of individual services leads directly to the requirement for hiding implementation details. Hiding implementation details such as timing leads to the use of persistent streams.

Existing messaging systems did not, however, support streaming microservices very well until recently. This prevented more than limited adoption of streaming microservices as a common architecture. This mirrors the way that early RPC mechanisms such as Java RMI and Corba were unsuitable for RPC-based microservices due to the way that changes in client or server were reflected through the remote call mechanism. True microservices could not really be built until more modern conventions for RPCs such as REST became widely accepted.

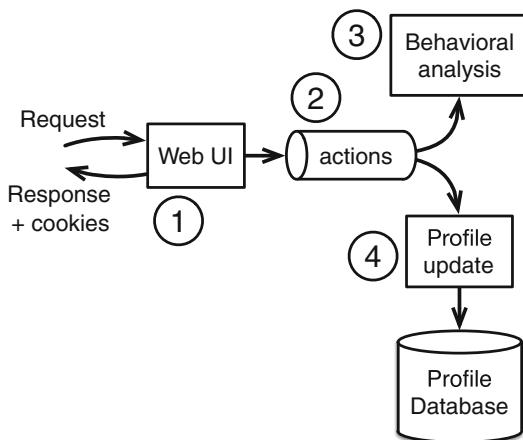
The problem with streaming microservices was that, until recently, message queuing systems put a premium on features that ensure that each message is processed exactly once. This exactly-once execution mirrors the way that a dataflow program executes. This mode of message consumption requires some kind of transaction per message and can severely limit the message rate and scalability (Videla and Williams, 2012; Snyder et al., 2011). More importantly, however, the transaction induces coupling between message

producers and consumers and between different consumers, which degrades the isolation necessary for real microservices.

The release of Apache Kafka (Narkhede et al., 2017) provided an alternative approach to messaging that has proven more fertile for microservices. With Kafka, neither producers nor consumers have any idea how many consumers there are, and messages are kept in order and are retained for a configurable (and typically relatively long) period of time with no regard to whether they have been processed. Consumers are responsible for maintaining a history of which messages they have processed, although the framework does have a provision for consumers to record their consumption point back into the stream. Kafka guarantees that all consumers will see messages in the same order within the same partition of a topic but does little else to support the context of consumers. This same model, and indeed, the same API, have been adopted by commercial vendors such as MapR in their streams technology.

Pattern: Do Some, Defer Some

In microservice-based systems, some work typically has to be done by a service to produce a response, but it is also common that a significant amount of work involved in the request can be deferred to another time, possibly when a larger batch of such deferred tasks can be executed at the same time. It is a rather natural step to put the data associated with these deferred tasks into a message stream. Each message specifies a unit of work to be done when the message is read from the stream. Such a do-some/defer-some pattern is illustrated in Fig. 1. Here, we have three microservices: one is the web UI (labeled 1 in the figure) implemented as a synchronous RPC-based microservice, and the other two (labeled 3 and 4 in the figure) are implemented as streaming microservices. The web UI can respond to requests as quickly as possible and can have a simpler implementation because it only needs to focus on the minimal amount of processing necessary to provide a response. By recording a summary of the request it just processed as an event in a message stream, the web UI enables



Streaming Microservices, Fig. 1 In a typical design of a web service, the web UI (1) is implemented as a conventional microservice that accepts requests and returns responses in a synchronous fashion. In addition, the web UI defers some work by putting an event into a message stream (2). At a later time, other microservices such as a behavioral analysis (3) or a profile database update (4) process these events. Using streaming microservices here avoids impact on the latency of the original web UI request and improves the isolation of all three services

other microservices to do additional processing outside the critical path of responding to web UI requests. Because the web UI doesn't even necessarily know of the existence of these other services, the implementation and operation of the web UI are highly decoupled from them. This decoupling means that each of the three services can be updated and redeployed independently of the other services.

In this example, the web UI doesn't need to wait for confirmation that the work of the other services has completed nor does it rely on the actual results of that work. All it needs is certainty that the deferred work has been persisted in the message stream and will get done eventually. In fact, the web UI can often be implemented with no knowledge of these other microservices. The decoupling benefits of not depending on direct results and relaxed temporal dependency suggest that connecting services with message streams is a useful alternative to the more widely known RPC-based connected microservices. This style of design was called “event sourcing” by Fowler in Fowler (2005) and later called “streaming

architecture” or “streaming microservices” (Dunning and Friedman, 2016).

One of the major advantages of streaming microservices over conventional RPC-based microservices is that streaming microservices allow temporal decoupling of services in addition to the ability to independently deploy and upgrade services. Temporal decoupling means that there is little dependency in time of computation for producers of messages and consumers. This decoupling allows major implementation details such as whether a producer or consumer runs intermittently or continuously to be hidden from other services. In the ultimate case of hiding implementation details, a consumer may not even have been implemented when messages are emitted by a producer. In that case, the producer cannot depend on the implementation details of the consumer since the consumer literally does not even exist when the message is produced by the producer.

Message Transport

The messaging technology used to communicate between streaming microservices has some key requirements that derive from the basic definition of microservices. Earlier messaging tools generally required a trade-off between performance and message persistence and induced coupling between producers and consumers, between producers on the same stream and between multiple consumers due to the semantics of the messaging system itself. These limitations made streaming microservices difficult to apply across a wide range of applications and thus substantially limited the uptake of streaming microservice architectures.

The effectiveness of the Kafka style of message streaming for microservices can seem a bit paradoxical; by providing fewer features and simpler semantics, it makes streaming microservices more practical. There are several major requirements that Kafka-esque systems satisfy, including:

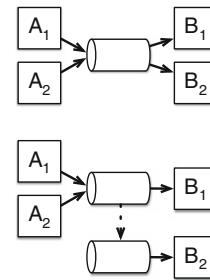
Persistence Messages are persisted in the message stream. This is logically required to enable service independence since there is no guarantee

that both producer and consumer are even running at the same time. Message persistence also improves failure recovery since a consumer can checkpoint input offsets and internal state, allowing clean restart on failure. Traditionally, however, persistence has been difficult to achieve at high performance levels.

Performance It is important that the message transport be fast enough for essentially any operational requirement. If this is not true and widely recognized, implementers will tend to proactively code around perceived performance risks. Wide adoption of streaming microservices can quickly lead to message throughputs in the millions of messages per second even at modest scale. It is not true, however, that all streaming microservices will require such high message rates. It is simply true that *some* applications will require such rates and using different streaming mechanisms for different applications or changing streaming technologies as services scale breaks down service independence because of a lack of pervasiveness.

Pervasive A key benefit of a microservices approach is that services can universally interoperate. For RPC oriented microservices, organizations typically standardize on a single RPC technology for this reason. For streaming microservices, similar standardization is required to gain adoption and allow interoperation of services. As such, any messaging system used has to meet technical requirements such as persistence and performance, but also social requirements like universal adoption within a set of services, that is, it must be pervasive. Put another way, if different services use different streaming technologies due to different performance levels, this introduces implementation coupling between systems that could be avoided with a single streaming system.

Ultimately, these properties of the underlying streaming technology result in a system that can allow services to function without knowledge of the implementation details of other services. In the upper part of Fig. 2, for example, services A_1 and A_2 can be producing data for B_1 . Service B_2 can then be brought into operation without



Streaming Microservices, Fig. 2 Examples of decoupling between processes. In the top example, producers A_1 and A_2 and consumers B_1 and B_2 operate independently and can be deployed without affecting any of the other services. In the bottom example, A_1 and A_2 produce data that is consumed by B_1 and B_2 from different replicas of the same stream. Producers and consumers in the two examples can't tell the difference between the two configurations

B_1 or either producer being aware. In the lower example, geographical separation can also be introduced by replicating the stream without any of the services being affected, except possibly by small differences in the latency for messages arriving at B_1 and B_2 .

Service Implementation

Streaming microservices can be implemented in a wide variety of ways. Particularly at the lower performance levels, it is possible to build reliable services using simple single-threaded code that reads units of work from a stream, performs the work and occasionally commits the current input point. This simple approach needs to be augmented with some sort of orchestrator such as Kubernetes that can restart or migrate the service as necessary. At slightly higher throughput levels, input streams can usually be partitioned to allow multiple processes to cooperate in a “consumer group.” By subscribing as part of the same consumer group, these processes will be assigned different partitions. The worker processes have to handle out-of-band messages notifying them of changes of partition control, but the basic structure is extremely simple, at least as long as the necessary processing on each unit of work is relatively simple.

As the required complexity of a service’s task goes up, it can be advantageous to use more

advanced systems to implement services. Options include Apache Spark's continuous streaming component (Apache Software Foundation, 2016b) or Apache Flink (Apache Software Foundation, 2016a; Friedman and Tzoumas, 2016). These more complex systems can handle complex joins and windowed transformations of streaming data while maintaining failure tolerance and high throughput.

Complementary Roles of the Message Stream and Database

In streaming microservices, message streams are used for communication between services, while databases are typically considered better for retaining state within a service. Often, a service will have a database that contains the latest state received via message stream. The sequence of states of such a database is, in some sense, equivalent to the messages in the stream. The primary difference is that the stream inherently retains the notion of time and all previous states but is difficult to probe for the latest value of any quantity. In contrast, the database makes probing for the latest value easy but loses information about previous states. As such, databases of this kind and the streams that update them can be seen as complementary views of the same data.

The virtue of storing state in a private database is that services can become decoupled in terms of time. If two consumers of a message stream update private databases in identical ways, there is no guarantee that at any given time that the two databases will have identical values because there is no guarantee that the two services will process messages at the same rate. We can guarantee, however, that both services will agree about the state of their databases relative to identical offsets in the message stream. This idea of equivalent, but not time-aligned, state is important because it allows much of the power of database transactions to be had without the abstraction destroying qualities of shared databases.

It should be noted that, as recommended by Fowler in his description of event sourcing, the messages in a message stream should generally be couched as heavily denormalized business-

level events rather than table level updates. The concept of "business-level event" is not particularly well defined, but it is very close to the concept of REST in that the message should contain everything that is needed to describe the event in a self-contained way. The purpose of using RESTful messages of this type is that it is relatively easy to translate such messages into corresponding low-level table updates, but it is much more difficult to translate a collection of table updates into business-level event descriptions. This means that using RESTful messages makes it so that different services can easily have different data models in their private databases, which makes their implementations more loosely coupled.

Key Applications

The streaming style of microservices has widespread application, especially associated with more back-end systems such as analytical systems in which throughput and aggregated analytical results at scale are particularly important (Dunning and Friedman, 2016). Streaming microservice applications often involve situations in which delayed processing is acceptable.

In general, situations in which the flexibility of a stream-based overall architecture is desired may benefit from a streaming microservices style of work. An example where a streaming microservices style approach would be particularly useful is in analysis of IoT (Internet of Things) sensor data.

S

Cross-References

- ▶ [Definition of Data Streams](#)
- ▶ [Rendezvous Architectures](#)

References

- Apache Software Foundation (2016a) Flink. <http://flink.apache.org>

- Apache Software Foundation (2016b) Spark structured streaming. <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>
- Dunning T, Friedman E (2016) Streaming architecture: new designs using Apache Kafka and MapR streams. O'Reilly Media, Inc., Sebastopol
- Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans Internet Technol 2(2):115–150. <http://doi.acm.org/10.1145/514183.514185>
- Fowler M (2005) https://martinfowler.com/eaaDev_EventSourcing.html
- Fowler M, Lewis J (2014) Microservices. <http://martinfowler.com/articles/microservices.html>
- Friedman E, Tzoumas K (2016) Introduction to Apache flink: stream processing for real time and beyond. O'Reilly Media, Inc., Sebastopol
- Hughes-Croucher T, Wilson M (2012) Node – up and running: scalable server-side code with javascript. O'Reilly. <http://shop.oreilly.com/product/0636920015956.do>
- Johnston WM, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages. ACM Comput Surv 36(1):1–34. <https://doi.org/10.1145/1013208.1013209>. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.7265>
- Narkhede N, Shapira G, Palino T (2017) Kafka – the definitive guide: real-time data and stream processing at scale. O'Reilly Media, Incorporated. <http://shop.oreilly.com/product/0636920044123.do>
- Newman S (2015) Building microservices. O'Reilly Media, Inc., <http://shop.oreilly.com/product/0636920033158.do>
- Snyder B, Bosanac D, Davies R (2011) ActiveMQ in action. Manning Publications Co., Greenwich. <https://www.manning.com/books/activemq-in-action>
- Videla A, Williams J (2012) RabbitMQ in action: distributed messaging for everyone. Manning Pubs Co series. Manning Publications Company. <https://www.manning.com/books/rabbitmq-in-action>

Streaming Process Discovery and Conformance Checking

Andrea Burattin
DTU Compute, Software Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

Synonyms

Online conformance checking; Online process discovery; Online process mining

Definitions

Streaming process discovery, streaming conformance checking, and streaming process mining in general (also known as *online process mining*) are disciplines which analyze event streams to extract a process model or to assess their conformance with respect to a given reference model. The main characteristic of this family of techniques is to analyze events immediately as they are generated (instead of storing them in a log for late processing). This allows to drastically reduce the latency among phases of the BPM lifecycle (cf. Dumas et al. 2013), thus allowing faster process adaptations and better executions.

Overview

A possible characterization of process mining algorithms is based on how they consume event data. Specifically, most of the algorithms focus on a (static) *event log*; however, there are algorithms which focus on *event streams*. An event log is a finite sampling of activities observed in a given time frame. An event stream, on the other hand, is an unbounded *sequence* of events, which contains events as they are executed.

Event streams, in fact, are specific types of data streams, where each data point refers to an event. General data streams have been investigated since many years, mainly to tackle problems such as frequency counting, classification, clustering, approximation, time series analysis, and change diagnosis (also known as novelty detection or concept drift detection) as summarized in Widmer and Kubat (1996), Gama (2010), Gaber et al. (2005), and Aggarwal (2007). To tackle these problems, it is possible to devise techniques in a data- or task-based orientation. *Data-based techniques* aim at extracting a significantly representative finite subset of the data stream, which is used for the analysis. *Task-based techniques*, on the other hand, adapt the computation to this new data modality, in order to minimize time and space complexity of the analysis.

The streams these algorithms have to deal with can be characterized based on their *operations model*. In particular, we might have:

- insert-only stream model (once an element is seen, it cannot be changed);
- insert-delete stream model (cf. n elements can be deleted or updated);
- additive stream model where seen item refers to numerical variables which are just incremented.

In the context of process mining, all available techniques assume the insert-only model: observations refer to activities which have been executed. The data mining literature, for example, in Golab and Özsu (2003) and Bifet et al. (2010), informally, defines data streams as a *fast sequence of data items*. However, in Bifet and Kirkby (2009), several assumptions on the data stream are reported, such as the following: the data is assumed to have a small and fixed number of attributes; the number of data points is very large; and the stream concepts (in the process mining context, the *concept* is the model underlying the events being generated) are assumed to be stationary or evolving. These assumptions make the processing of data streams very different from conventional relational models. Specifically, as detailed in Gama (2010, Table 2.1), the data elements arrive online, with no control by the system, in an unbounded amount. This imposes the analysis to be incremental: once an element is received, it is discarded or analyzed. If it is

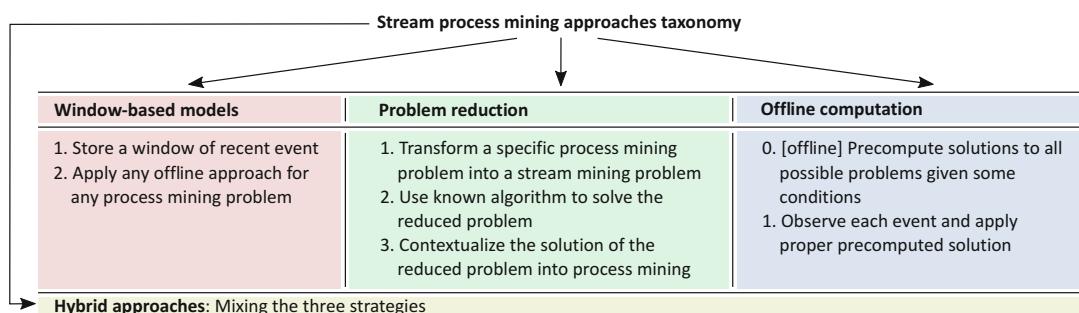
analyzed, it cannot be explicitly retrieved again afterward (i.e., its information is aggregated and summarized). Additionally, to cope with concept drifts, old observations should be replaced by new ones, and it is not possible to have one-time queries but a continuous “querying mechanism” is necessary.

Key Research Findings

This section provides some general ideas on how to tackle the problem of process mining from an event stream. Three general strategies are sketched, and in the upcoming subsections, details regarding actual instantiations of the ideas are reported.

Figure 1 depicts a taxonomy of the different approaches investigated so far. In this text, we will not focus on hybrid approaches: they are characterized by very heterogeneous solutions and therefore there is no proper generalization possible.

Window models. In order to perform process mining on a stream of event, the simplest approach is to devise a data-based technique to store only the set of most recent events observed and periodically analyze them. Whenever there is no more memory available, the oldest event is discarded. This approach, called *window model*, is described in Algorithm 1. The algorithm enters an endless loop where, at each iteration, a new event is observed. Then, the system checks whether it is necessary to remove old events or not, and



Streaming Process Discovery and Conformance Checking, Fig. 1 Taxonomy of the different approaches to solve the different stream process mining problems. For each technique, corresponding general steps are sketched

Algorithm 1: Window model

Input: S : event stream
 M : memory model
 \max_M : maximum memory
 A : additional information (e.g., a reference model), can be \emptyset

```

1 forever do
2   // Observe a new event
3    $e \leftarrow \text{observe}(S)$ 
4   // Memory update
5   if  $\max(M) \geq \max_M$  then
6     |  $\text{dequeue}(M)$  // Forgetting
7   end
8    $\text{insert}(M, e)$ 
9   // Mining update
10  if  $\text{perform mining}$  then
11    | // Memory into event log
12    |  $L \leftarrow \text{convert}(M)$ 
13    |  $\text{ProcessMining}(L, A)$ 
14  end
15 end

```

then the new event is inserted. Periodically, the system converts the memory into a standard event log, and classical process mining algorithms are applied on it. The literature, in Babcock et al. (2002), identifies at least two memory models capable of storing event: *sequence based* and *timestamp based*. The first approach (which is typically implemented with *sliding windows*) consists of a FIFO queue of fixed size. The observed events are stored in the window, and once the maximum capacity is reached, the oldest event is removed. In a timestamp-based window model, the approach is very similar, but the memory size is not fixed. Instead, the removal is based on the “age” of the observation: the memory keeps only the events observed within the given time span.

This approach comes with several advantages, such as the possibility to reuse every mining algorithm already available for event logs. However, the memory management is extremely problematic and has a huge impact on the performance of the approach. Specifically, all window-based approaches have poor summarizing capabilities since, for example, duplicate events require two “memory slots,” even though they may not provide new information.

Algorithm 2: Lossy counting

Input: S : data stream
 ϵ : maximal approximation error

```

1  $T \leftarrow \emptyset$  // Initially empty set
2  $N \leftarrow 1$  // Number of observed events
3  $w \leftarrow \lceil \frac{1}{\epsilon} \rceil$  // Bucket width
4 forever do
5   |  $e \leftarrow \text{observe}(S)$ 
6   |  $b_{curr} \leftarrow \lceil \frac{N}{w} \rceil$ 
7   | /* Is there a tuple in  $T$  with  $e$  as
8   |   first component? */ *
9   | if  $e$  is already in  $T$  then
10  |   | Increment the frequency of  $e$  in  $T$ 
11  | else
12  |   | Insert  $(e, 1, b_{curr} - 1)$  in  $T$ 
13  | end
14  | if  $N \bmod w = 0$  then
15  |   | forall the  $(a, f, \Delta) \in T$  s.t.  $f + \Delta \leq b_{curr}$  do
16  |   |   | Remove  $(a, f, \Delta)$  from  $T$ 
17  | end
18 end

```

Problem reduction. The second possible way of tackling the streaming process mining problems is to employ a task-based technique. For example, it is possible to *reduce* the process mining problem to another well-established problem and therefore reuse algorithms specifically devised and optimized for the necessity at hand. Of course, it is also possible to devise a completely new algorithm specifically tailored to solve the given situation. This approach, for example, has been used to reduce the problem of streaming process discovery to the *frequency counting problem*. This problem consists of counting the frequencies of given variables over a stream. In order to reduce the process discovery to such problem, it is important to understand *what is* a variable in the process mining context and whether it is possible to identify it.

An example of efficient algorithm for the approximated frequency counting problem is Lossy Counting, here summarized in Algorithm 2 and described in Manku and Motwani (2002). The idea is to conceptually split the observed stream in buckets, each with a fixed size. The approach takes as input the stream and the maximal approximation error on the counting $\epsilon \in [0, 1]$, which drives the size of each bucket. The approach stores entries in a data structure T where each

component (e, f, Δ) provides the element e of the stream, the estimated frequency for it f , and the maximum number of times it could have occurred Δ . When a new event is observed in the stream, the algorithm checks if it is already in T and, in case, it increments its counter f by one. Otherwise a new entry in T is created. Periodically (i.e., every time a new conceptual bucket starts), the algorithm cleans the memory, by removing elements not frequently observed. This algorithm has no memory bound. Specifically, the size of the data structure T depends on the stream and on the approximation error. However, a variation of the algorithm to enforce fixed amount of memory is described in Da San Martino et al. (2012).

The most relevant benefit of the problem reduction approach is that, once the process mining problem has been reduced to the new one, it is possible to use algorithms already devised for the reduced problem. However, such reduction might not be trivial and all assumptions of the used approach have to be met.

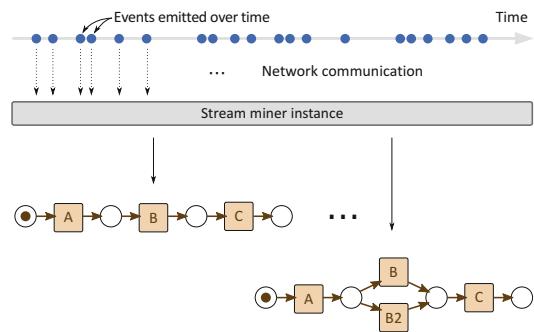
Offline computation. The last approach we present consists of moving the computation of the solutions to the given problem from online setting to offline. In other words, the idea is to identify and solve all subproblems the stream may provide. Adopting this approach will help us in dealing with all situations we are going to observe, still keeping the complexity of the online processing constant.

The advantage of this approach is the possibility of having extremely expensive solutions “cached” in advance which are then just reused whenever needed. Though there are several drawbacks, it is not possible to apply this approach to all online process mining problems. Additionally, by computing everything in advance, we lose the possibility of adapting the precomputed solutions to the contextual information, which might be uniquely specific to the running process instance.

In the upcoming subsections, one example of each technique will be presented and detailed by presenting two process mining activities.

Process Discovery

The first problem we present is the online process discovery. A graphical conceptualization of the



Streaming Process Discovery and Conformance Checking. Fig. 2 Conceptualization of the streaming process discovery idea as in Burattin et al. (2014b)

problem is reported in Fig. 2. The idea is to have a source which is generating an event stream. This event stream is consumed by a miner which generates a representation of the underlying process model and keeps it up-to-date with respect to the observed behavior.

The first approach available to tackle this problem is reported in Burattin et al. (2012, 2014b). In their works, authors employ a problem reduction strategy. Specifically, they reduce the Heuristics Miner algorithm, described in van der Aalst and Weijters (2003), to the frequency counting problem. To do that, they assume *direct following relationships* as variables, and by counting them, they are able to use the Heuristics Miner’s metric to reconstruct the high-level business patterns in terms of Heuristics Net or Petri Net. Authors test different algorithms to solve the frequency counting problem, such as Lossy Counting and Lossy Counting with Budget. As baseline approach, authors use a sliding window mechanism where Heuristics Miner is iteratively applied. The sliding window approach is constantly outperformed by other approaches which provide a better usage of the available resources. A similar approach, called StrProM, tracks the direct following relationships. This approach, presented in Hassani et al. (2015), keeps an updated prefix tree by deriving a solution based on Lossy Counting with Budget. The direct following relationships are then used to construct a process model using the set of rules of Heuristics Miner.

As for the previously mentioned approaches, in Maggi et al. (2013) and Burattin et al. (2014a, 2015), authors investigate the problem of discovering a process represented in Declare (cf. Pesic et al. 2007). Specifically, their idea is to instantiate several “replayers,” one for each Declare template to mine. Then, specific behavior for each template is implemented. To keep track of the replay statuses, authors use Lossy Counting-based strategies. Authors apply sliding window as baseline, and again, the performances of the Lossy Counting strategies are better with respect to the simple application of offline approaches over a sliding window.

In Redlich et al. (2014b), authors present the adaptation of CCM to cope with event streams. The basic idea of CCM (cf. Redlich et al. 2014a) is to identify subsequences of events in order to identify footprints of specific process patterns. In CCM, relevant patters and corresponding footprints are described. Aging factors are employed to the collected information in order to give more importance to recent behavior.

The most recent work tackling the online process discovery is reported in van Zelst et al. (2017b). In their paper, authors generalize previously instantiated concepts: they present an architecture, namely, S-BAR, which keeps an updated abstract representation of the stream (e.g., direct follow relationships), and they use it as starting point to infer an actual process model. In their work, authors present the adaptations of different mining algorithms: α (cf. van der Aalst et al. 2004), Heuristics Miner (cf. van der Aalst and Weijters 2003), and Inductive Miner (cf. Leemans et al. 2013). To show the generability of their abstract representation, authors also show a miner based on region theory (cf. van der Aalst et al. 2008). In order to keep their abstraction updated, authors reduce their problem to frequency counting, thus using Lossy Counting, Space Saving (cf. Metwally et al. 2005), and Frequent (cf. Karp et al. 2003).

Conformance Checking

The problem of online conformance checking has received attention in the declarative domain. In this case, it used to be called *operational support*,

and its aim is to understand whether a set of constraints is being violated or not. To achieve that, in Maggi et al. (2011, 2012), authors devise an approach to represent the behavior as an automaton and executions are replayed on it. Additionally, each process instance is labeled with one of four possible fulfillment states: permanently/temporarily violated/fulfilled.

Concerning imperative models, online conformance checking received less attention. At present time, only two approaches have been specifically designed to tackle the online conformance checking problem. One approach, described in van Zelst et al. (2017a), aims at reusing the concept of *alignment* in order to compute the optimal alignment just for the prefix of the trace seen up to a given point in time. To this end, authors first devise a technique to compute prefix alignments. Authors prove the optimality of the prefix alignment they discover. Up to this point, their approach is incremental but not really online (i.e., it is able to work on partial cumulative information, but in theory they need infinite memory to backtrack in order to find the optimal alignment). To solve this issue, authors mention the possibility to implement memory management, either falling into a window-based model or as problem reduction. Authors, however, do not implement or test either memory management approach. However, they test their technique against different number of backtracking steps required to find the prefix alignment.

Another conformance checking approach belongs to the offline computation category and is described in Burattin and Carmona (2017) and Burattin (2017). In this case, given a Petri Net as input, authors describe a technique to elicit a transition system containing all possible transitions from one marking to another one, even those which are not described by the original model. Each transition in this elicited model is then associated with a cost. Transitions allowed by the original Petri Net have cost 0; all others have cost > 0 . This way, by replaying a trace on such transition system and summing the costs, authors show that conformant traces will have cost 0 and non-conformant trace always have cost larger than 0. Additionally, authors ensure the

determinism of such transition system and the possibility, from each state, to have one transition for each possible activity. These two last properties guarantee the suitability of the approach for online settings.

Another conformance checking approach is presented in Weber et al. (2015). In this case, authors propose a RESTful service which performs a token replay on a BPMN model. In particular, authors implemented a *token pull mechanism*. Authors refer this approach as online primarily because of its interface, while no explicit guarantee is mentioned in terms of memory usage and computational complexity.

Other Applications

Online process mining has been applied also to discover cooperative structures out of event streams. For example, in van Zelst et al. (2016), authors are able to process an event stream and update the set of relationships of a cooperative resource network.

Additionally, in order to conduct research on stream process mining, it is useful to simulate an event stream for which we know the actual original source. To achieve that, mainly two approaches are available. The first is a tool called PLG2 (<http://plg.processmining.it/>), described in Burattin (2016). It allows to generate a random model or to load a BPMN file and stream events referring to actual executions. The stream is sent over a TCP/IP connection, and the tool allows different configurations in terms of noise and concept drift. The second tool is described in van Zelst et al. (2015). This tool allows researchers to design a model – as Petri Net – in CPN (<http://www.cpntools.org/>) and then import it into ProM (<http://www.promtools.org/>) where the XSEventStream Publisher plugin can be used to simulate a stream. Please note that, in this case, the stream exists just within ProM.

Key Applications

Due to the novelty of the topic, we are not aware of any deployment of streaming process min-

ing techniques in real settings. However, more generally, online process mining techniques are relevant in all cases where it is important to have results in real time, to immediately enact proper responses. In this section, examples related to IT setting will be provided, but business-oriented applications are absolutely possible and relevant as well.

Online process discovery might be useful in settings where it is important to analyze immediately the behavior of the system. For example, reconstructing the behavior of the services used in a website might be useful in order to see what is currently under stress and what is going to be used afterward. Exploiting this information could improve the resource allocation (e.g., upscaling or downscaling the server capacity on the fly).

Online conformance checking is also useful whenever it is important to immediately detect deviations from reference behavior to enact proper countermeasures. For example, the kernel of an operating system exposes some services for applications. These services should be combined in some specific ways (e.g., a file should be `open()` and then either `write()` or `read()` or both appear, and eventually the file should be `close()`) which represent the reference behavior. If an application is observed strongly violating such behavior, it might be an indication of strange activities going on, for example, in order to bypass some imposed limitations or privileges.

S

Future Directions for Research

As previously mentioned, online process mining is a relatively new area of investigation. Some techniques are available for the discovery of the process (both as imperative and declarative models). Very recently, online conformance checking also received attention, but several improvements are still needed.

First of all, techniques should improve their efficiency, for example, in terms of memory consumption. This represents an important research direction since, in the online setting, the amount of available memory is fixed, thus representing a

key resource. To tackle this problem, it is necessary to work on the summarization capabilities used by the algorithms in order to find more compact ways of storing the same information.

Related to the previous point is the quality and quantity of information, and contextual data algorithms are able to extract out of the same event stream. For example, a process discovery algorithm might be extended to extract more complex control flow patterns, or the algorithm might be modified to return not just the result but the confidence on the provided outcomes.

Additionally, there are more technical issues that algorithms should be able to cope with, for example, dealing with a stream where the arrival time of events does not coincide with their actual execution. In this case, it would be necessary to reorder the list of events belonging to the same process instance before mining them. Please note that assuming different orders implies a sort of different element models of the stream (i.e., it becomes an “insert-delete stream model,” where the order of events can change). Another relevant issue might be the inference of the termination of a process instance.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Conformance Checking](#)
- ▶ [Declarative Process Mining](#)
- ▶ [Definition of Data Streams](#)
- ▶ [Introduction to Stream Processing Algorithms](#)

References

- Aggarwal CC (2007) Data streams: models and algorithms. *Advances in database systems*. Springer, Boston. <https://doi.org/10.1007/978-0-387-47534-9>
- Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, pp 1–16. <https://doi.org/10.1145/543614.543615>
- Bifet A, Kirkby R (2009) Data stream mining: a practical approach. Technical report, Centre for open software innovation – The University of Waikato
- Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis learning examples. *J Mach Learn Res* 11:1601–1604
- Burattin A (2016) PLG2 : Multiperspective process randomization with online and offline simulations. In: Online proceedings of the BPM Demo Track 2016, CEUR-WS.org, vol 1789, pp 1–6
- Burattin A (2017) Online conformance checking for petri nets and event streams. In: CEUR Workshop Proceedings, vol 1920
- Burattin A, Carmona J (2017, in press) A framework for online conformance checking. In: Proceedings of the 13th international workshop on business process intelligence (BPI 2017). Springer
- Burattin A, Sperduti A, van der Aalst WM (2012) Heuristics miners for streaming event data. ArXiv CoRR <http://arxiv.org/abs/1212.6383>
- Burattin A, Maggi FM, Cimitile M (2014a) Lights, camera, action! business process movies for online process discovery. In: Proceedings of the 3rd international workshop on theory and applications of process visualization (TAPoViz 2014)
- Burattin A, Sperduti A, van der Aalst WM (2014b) Control-flow discovery from event streams. In: Proceedings of the IEEE congress on evolutionary computation. IEEE, pp 2420–2427. <https://doi.org/10.1109/CEC.2014.6900341>
- Burattin A, Cimitile M, Maggi FM, Sperduti A (2015) Online discovery of declarative process models from event streams. *IEEE Trans Serv Comput* 8(6):833–846. <https://doi.org/10.1109/TSC.2015.2459703>
- Da San Martino G, Navarin N, Sperduti A (2012) A lossy counting based approach for learning on streams of graphs on a budget. In: Proceedings of the twenty-third international joint conference on artificial intelligence. AAAI Press, pp 1294–13010
- Dumas M, La Rosa M, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer
- Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. *ACM Sigmod Rec* 34(2):18–26. <https://doi.org/10.1.1.80.798>
- Gama J (2010) Knowledge discovery from data streams. Chapman and Hall/CRC, Boca Raton. <https://doi.org/10.1201/EBK1439826119>
- Golab L, Özsu MT (2003) Issues in data stream management. *ACM SIGMOD Rec* 32(2):5–14. <https://doi.org/10.1145/776985.776986>
- Hassani M, Siccha S, Richter F, Seidl T (2015) Efficient process discovery from event streams using sequential pattern mining. In: 2015 IEEE symposium series on computational intelligence, pp 1366–1373. <https://doi.org/10.1109/SSCI.2015.195>
- Karp RM, Shenker S, Papadimitriou CH (2003) A simple algorithm for finding frequent elements in streams and bags. *ACM Trans Database Syst* 28(1):51–55. <https://doi.org/10.1145/762471.762473>
- Leemans SJ, Fahland D, van der Aalst WM (2013) Discovering block-structured process models from event logs – a constructive approach. In: Proceedings of Petri

- nets. Springer, Berlin/Heidelberg, pp 311–329. https://doi.org/10.1007/978-3-642-38697-8_17
- Maggi FM, Montali M, Westergaard M, van der Aalst WM (2011) Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Proceedings of the 9th international conference on business process management. Springer, Berlin/Heidelberg, pp 132–147. https://doi.org/10.1007/978-3-642-23059-2_13
- Maggi FM, Montali M, van der Aalst WM (2012) An operational decision support framework for monitoring business constraints. In: Proceedings of 15th international conference on fundamental approaches to software engineering (FASE), pp 146–162. https://doi.org/10.1007/978-3-642-28872-2_11
- Maggi FM, Bose RPJC, van der Aalst WM (2013) A knowledge-based integrated approach for discovering and repairing declare maps. In: 25th international conference, CAiSE 2013, 17–21 June 2013. Springer, Berlin/Heidelberg/Valencia, pp 433–448. https://doi.org/10.1007/978-3-642-38709-8_28
- Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of international conference on very large data bases. Morgan Kaufmann, Hong Kong, pp 346–357
- Metwally A, Agrawal D, Abbadi AE (2005) Efficient computation of frequent and Top-k elements in data streams. In: Database theory – ICDT 2005. Springer, Berlin/Heidelberg, pp 398–412. https://doi.org/10.1007/978-3-540-30570-5_27
- Pesic M, Schonenberg H, van der Aalst WM (2007) DE-CLARE: full support for loosely-structured processes. In: Proceedings of EDOC. IEEE, pp 287–298. <https://doi.org/10.1109/EDOC.2007.14>
- Redlich D, Molka T, Gilani W, Blair G, Rashid A (2014a) Constructs competition miner: process control-flow discovery of BP-domain constructs. In: Proceedings of BPM 2014, pp 134–150. https://doi.org/10.1007/978-3-319-10172-9_9
- Redlich D, Molka T, Gilani W, Blair G, Rashid A (2014b) Scalable dynamic business process discovery with the constructs competition miner. In: Proceedings of the 4th international symposium on data-driven process discovery and analysis (SIMPDA 2014), vol 1293, pp 91–107
- van der Aalst WM, Weijters TAJMM (2003) Rediscovering workflow models from event-based data using little thumb. *Integr Comput Aided Eng* 10(2):151–162
- van der Aalst WM, Weijters TAJMM, Maruster L (2004) Workflow mining: discovering process models from event logs. *IEEE Trans Knowl Data Eng* 16:2004
- van der Aalst WM, Günther CW, Rubin V, Verbeek EHMW, Kindler E, van Dongen B (2008) Process mining: a two-step approach to balance between underfitting and overfitting. *Softw Syst Model* 9(1):87–111. <https://doi.org/10.1007/s10270-008-0106-z>
- van Zelst SJ, van Dongen B, van der Aalst WM (2015) Know What you stream: generating event streams from CPN models in ProM 6. In: CEUR workshop proceedings, pp 85–89
- van Zelst SJ, van Dongen B, van der Aalst WM (2016) Online discovery of cooperative structures in business processes. In: Proceedings of the OTM 2016 conferences. Springer, pp 210–228
- van Zelst SJ, Bolt A, Hassani M, van Dongen B, van der Aalst WM (2017a) Online conformance checking: relating event streams to process models using prefix-alignments. *Int J Data Sci Anal*. <https://doi.org/10.1007/s41060-017-0078-6>
- van Zelst SJ, van Dongen B, van der Aalst WM (2017b) Event stream-based process discovery using abstract representations. *Knowl Inform Syst* pp 1–29. <https://doi.org/10.1007/s10115-017-1060-2>
- Weber I, Rogge-Solti A, Li C, Mendling J (2015) CCaaS: online conformance checking as a service. In: Proceedings of the BPM demo session 2015, vol 1418, pp 45–49
- Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. *Mach Learn* 23(1):69–101. <https://doi.org/10.1007/BF00116900>

Streaming SQL Queries

► Continuous Queries

StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing

André Martin¹, Andrey Brito², and Christof Fetzer¹

¹TU Dresden, Dresden, Germany

²UFCG, Campina Grande, Brazil

S

Introduction

During the past decade, we have been witnessing a massive growth of data. In particular the advent of new mobile devices such as smartphones, tablets and online services like Facebook and Twitter created a complete new era for data processing. Although there exist already well-established approaches such as MapReduce (Dean and Ghemawat, 2008) and its open-source implementation Hadoop (2015) in order to cope with these large amounts of data, there is a recent trend of moving

away from batch processing to low-latency online processing using event stream processing (ESP) systems. Inspired by the simplicity of the MapReduce programming paradigm, a number of open-source as well as commercial ESP systems have evolved over the past years such as Apache S4 (Neumeyer et al., 2010; Apache, 2015) (originally pushed by Yahoo!), Apache Storm (2015) (Twitter), and Apache Samza (2015) (LinkedIn), addressing the strong need for data processing in near real time.

Since the amount of data being processed often exceeds the processing power of a single machine, ESP systems are often carried out as *distributed systems* where multiple nodes perform data processing in a cooperative manner. However, with an increasing number of nodes used, the probability for a failure increases which can lead either to partial or even full system outages. Although several well-established approaches to cope with system failures for distributed systems are known in literature, providing fault tolerance for ESP systems is challenging as those systems operate on constantly flowing data where the input stream cannot be simply stopped and restarted during system recovery.

One possibility for providing fault tolerance in ESP systems is the usage of checkpointing and logging, also known as *rollback recovery-passive replication* in literature where the state of an operator is periodically checkpointed to some fault-tolerant stable storage and so-called in-flight events are kept in in-memory logs at upstream nodes (upstream backup) (Hwang et al., 2005; Gu et al., 2009). During system recovery, the most recent checkpoint is being loaded, and previously in-flight events are replayed. An alternative to rollback recovery is *active replication* (Schneider, 1990; Martin et al., 2011a) where two identical copies of an operator are deployed on different nodes performing redundant processing. If one of the two copies crashes, the system continues to operate without having to initiate a long recovery procedure as in passive replication.

Although active replication provides the quickest recovery, it requires almost twice the resources, while passive replication consumes only little resources; however, it suffers from

long recovery times. Despite the fact that both fault tolerance approaches have different characteristics with regard to recovery times and resource overhead, both require a *deterministic* processing of events. Deterministic processing ensures that all replicas process events in the same predefined order which is important in order (i) to reliably filter out duplicated events when using active replication and (ii) to provide reproducibility of events needed in order to recover precisely (i.e., neither loosing any events nor processing events twice) using passive replication. However, the use of deterministic execution imposes a non-negligible overhead as it increases processing latency and lowers throughput at the same time due to the cost of event ordering.

Key Research Findings

This chapter presents three approaches to reduce the overhead imposed by fault tolerance in ESP systems. They are presented in the following order: First,

1. the architecture and implementation of STREAMMINE3G are discussed, an ESP system that was built from scratch to study and evaluate novel fault tolerance and elasticity mechanisms,
2. an algorithm to reduce the overhead imposed by deterministic execution targeting commutative tumbling windowed operators and improving the throughput by several orders of magnitude when used with passive and active replication,
3. an approach to improve the overall system availability by utilizing spare but paid cloud resources, and
4. an adaption-based approach that minimizes the operational costs by selecting the least expensive fault tolerance scheme at runtime based on user-provided constraints.

Roadmap section “[STREAMMINE3G Approach](#)” introduces the reader to the architecture and implementation of STREAMMINE3G, the ESP system used for evaluating the proposed approaches. In section “[Lowering Runtime Overhead for Pas-](#)

sive Replication”, an approach is presented that lowers the overhead of event ordering by introducing the notion of an *epoch* and evaluated it for the use with passive replication. An extension of this approach is discussed in section “Lowering Runtime Overhead for Active Replication” for the use with active replication by proposing an epoch-based merge algorithm and a lightweight consensus protocol. Next, section “Improving Resource Utilization and Availability Through Active Replication” details how to improve system availability by using a hybrid approach of passive standby and active replication by utilizing spare but paid cloud resources, while in section “Adaptive and Low-Cost Fault Tolerance for Cloud Environments”, an adaptation approach for fault tolerance is presented that allows to lower the overall resource consumption while still satisfying user-specified constraints such as recovery time and recovery semantics. Finally, the chapter concludes in section “Conclusions” with a short summary of the approaches and the contributions.

STREAMMINE3G Approach

The following section provides a brief overview about STREAMMINE3G, the ESP system used to implement and evaluate the proposed approaches.

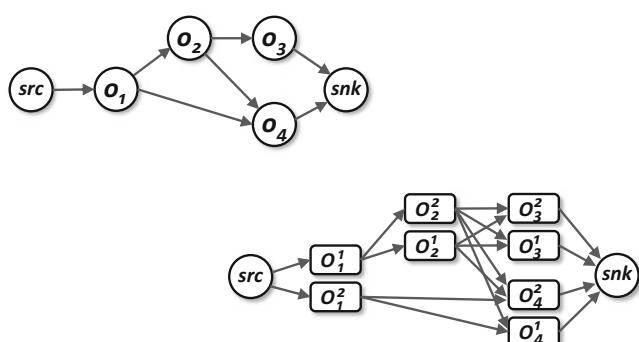
STREAMMINE3G is a highly scalable ESP system targeting low-latency data processing of streaming data. In order to analyze data, users can either opt for writing their own custom operators using the provided MapReduce-like interface and implementing a user-defined function (UDF) or

choose from an existing set of standard complex event processing (CEP) operators such as filter, join, aggregation, and others. In addition to the operators, users must specify the order events are supposed to traverse the previously selected operators using a topology. A topology in STREAMMINE3G is represented by a directed acyclic graph (DAG) where each vertex, i.e., an operator, can have multiple upstream and downstream operators as shown in Fig. 1 (left). In order to achieve scalability, operators in STREAMMINE3G are partitioned as depicted in Fig. 1 (right). Each partition processes only a subset of events from the incoming data stream. For data partitioning, users can either implement their own custom partitioner as in MapReduce or use the provided hash-based or key-range-based partitioner.

A typical STREAMMINE3G cluster consists of several nodes where each node runs a single STREAMMINE3G process hosting an arbitrary number of operator partitions, named slices as shown in Fig. 2. One of such nodes takes up the role of a manager which is responsible for placing operator partitions across the set of available nodes as well as moving partitions (through a migration mechanism) to other nodes for load balancing in situations of overload or underutilization. An overload can be detected by the manager node by analyzing the system utilization of each node, which is periodically reported through heartbeat messages exchanged between nodes.

In order to prevent the node hosting the manager component being the single point of failure, the state of the component is stored in Zookeeper (Hunt et al., 2010) upon each

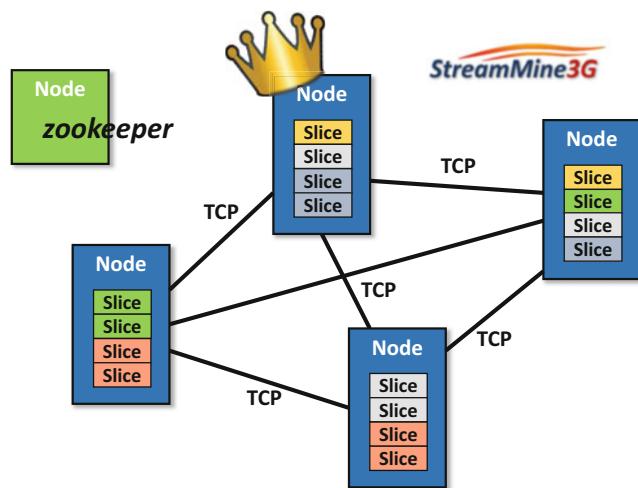
StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing,
Fig. 1 Logical (top left) and physical (bottom right) representation (with two partitions for each operator) of a query graph



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing,

Fig. 2 A

STREAMMINE3G cluster. One node acts as a master/manager, while others serve as workers hosting an arbitrary number of slices (operator partitions)



reconfiguration of the system. In the event of a crash of the node, another node can transparently take over the role of the manager by simply recovering with the previously persisted state.

Lastly, STREAMMINE3G supports the implementation of stateless and stateful operators. However, contrary to other ESP systems such as Apache S4 and Storm that have either no, or only limited, state support, STREAMMINE3G offers an explicit state management interface to its users. The interface frees the users from the burden of having to implement their own bridle locking mechanism to ensure consistent state modifications when processing events concurrently (to exploit multiple cores) and provides a full stack of mechanisms for state checkpoints, recovery, and operator migration. In order to use these mechanisms, users are only required to implement appropriate methods for serialization and de-serialization of the state that can comprise arbitrary data structures.

Lowering Runtime Overhead for Passive Replication

In the following section, an approach for lowering the overhead for passive replication by introducing the notion of an epoch is presented.

Introduction and Motivation

ESP applications are often long-running operations that continuously analyze data in order to carry out some form of service. In order to identify patterns and correlations between consecutive events, operators are often implemented as stateful components. One way for providing fault tolerance for such components is to combine periodic checkpointing with event logging for a later replay. However, since events may arrive in different orders at an operator during a recovery than they would have arrived originally due to small delays in network packet delivery, the immediate processing of such events would lead to possibly inconsistent results. One way of preventing such situations is to *order events* prior to the processing in order to ensure a deterministic input to the operator code at all times. However, the ordering of events is costly as it introduces latency and lowers throughput as shown later.

Fortunately, many ESP operators used in ESP applications share the property of *commutativity* and operate on jumping windows where the order of processing within such windows is irrelevant for the computation of the correct result. Examples for such operators are joins and aggregations. However, processing the same event twice or even missing an event may still distort the result of those operators. Hence, determinism is still needed in order to provide *exactly once processing semantics*.

Approach

Based on the observation that many operators are commutative and operate on jumping windows, the notion of an *epoch* is introduced. An epoch comprises a set of events based on their timestamps and matches the length of the window an operator is working on. In order to assign events correctly to those epochs, i.e., the time-based windows, events are equipped with monotonically increasing timestamps. The key idea of the approach is to process events within such epochs, i.e., windows in *arbitrary order*, however, *processing epochs itself in order*. Exactly once semantics can now be provided by solely performing checkpointing and recovery at *epoch boundaries* as at those points in time the system still provides a deterministic snapshot.

Evaluation and Results

For the evaluation of the approach, a canonical streaming word count application has been implemented that consists of two operators: a stateless map operator that splits lines read from a Wikipedia corpus file into individual words which are then accumulated by a stateful reduce operator. The stateful operator summarizes the word frequencies using a jumping window, i.e., an epoch where the length of the epoch is defined in terms of file position the word originated from in the source file. The performance of the epoch-based approach has been evaluated by comparing

it with an execution that does not perform any event ordering and an execution that performs a strict event ordering. In strict ordering, every single event is ordered rather than applying the weak ordering scheme based on epochs. The results of the measurements are shown in Fig. 3.

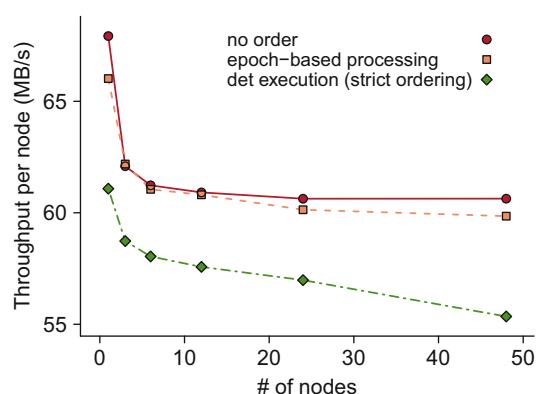
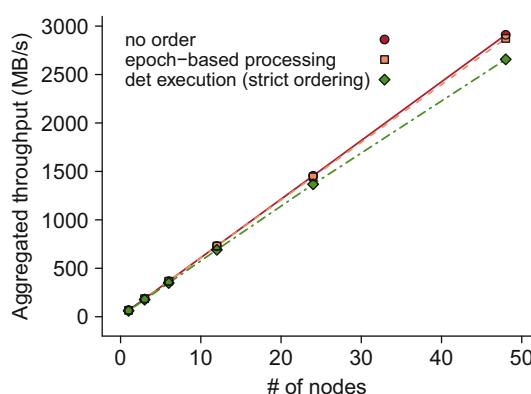
Figure 3 (left) shows the accumulated throughput for the experiment running on a 50-node cluster, while Fig. 3 (right) depicts the per node throughput. The experiments reveal that using the proposed weak ordering scheme, a similar throughput as when not applying any ordering can be achieved, however, providing precise recovery semantics as when using strict ordering, i.e., deterministic execution.

Lowering Runtime Overhead for Active Replication

In the previous section, an approach for reducing the overhead of event ordering was presented that provides *exactly once processing semantics* and *precise recovery* when used with passive replication which is traditionally based on checkpointing and in-memory event logging. In this section, an extension will be presented that can be used with active replication.

Introduction and Motivation

Active replication is a useful approach to recover applications that accumulate large portions of state as the secondary instance is holding the



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 3 Aggregated (left) and per node throughput (right) with increasing number of

nodes. The epoch-based approach significantly outperforms deterministic execution

state already in memory rather than reading it from disk during a recovery. However, in order to use active replication, a costly atomic broadcast or deterministic execution (i.e., strict event ordering) is needed in order to ensure consistent processing across all replicas. However, when using commutative and windowed operators, event ordering solely serves the purpose of maintaining consistency across replicas but does not have any impact on correctness due to the commutativity.

Approach

Inspired by the previous epoch-based processing approach, the following approach can be considered as an extension as it performs an *epoch-based deterministic merge* that ensures correctness for active replication, however, at a much lower cost than a strict event order/merge. The key idea of the approach is to merge epochs rather than individual events which is far less costly than a strict per event merge as shown in the evaluation below.

In order to perform an epoch-based deterministic merge, events arriving from different upstream channels are enqueued on a per epoch basis in separate so-called *epoch bags* first. Once an epoch completes, i.e., all channels have passed the epoch boundary, the content of the epoch bags is merged by processing the bags in the order of predefined channel identifiers. Since the channel identifiers are globally defined and events from upstream operators are delivered in FIFO order

through TCP, the final sequence of events is identical and deterministic for all replicas.

In case upstream operator replicas or sources deliver identical but differently ordered sequences of events and in order to reduce latency caused by stragglers, a *lightweight consensus protocol* can be used that selects for the available upstream replicas the set of bags to merge so that all downstream peers process the same sets of events. The protocol stops furthermore the propagation of non-determinism while decreasing latency at the same time.

Evaluation and Results

For the experimental evaluation of the approach, a canonical application operating on jumping windows was implemented that consists of three operators, a partitioned source operator, an equijoin that combines the output from the source operator, and a sink. In order to assess the performance of the approach, it was compared with an out-of-order execution, a strict ordering, the epoch-based merge, and the consensus-based protocol. The outcome of the experiments is depicted in Fig. 4.

Figure 4 (left) depicts the accumulated throughput for the experiment running on a 50-node cluster, while Fig. 4 (right) depicts the per node throughput. As shown in the figures, the epoch-based deterministic merge has a noticeable higher throughput than strict determinism (ordering), while there is only a

# of nodes	det execution (strict ordering)	no order	no order no duplicates	epoch-based processing	epoch-based det merge	epoch-based det merge + con
1	~100	~100	~100	~100	~100	~100
12	~200	~400	~400	~400	~400	~400
24	~400	~800	~800	~800	~800	~800
48	~800	~1600	~1600	~1600	~1600	~1600

StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 4 Horizontal scaling – aggregated (left) and per node (right) throughput with increasing number of nodes

marginal difference for the consensus-based variant in comparison to the epoch-based deterministic merge without agreement.

Improving Resource Utilization and Availability Through Active Replication

While the objective of the previously presented approaches was to minimize the runtime overhead for fault tolerance by introducing a weak ordering scheme based on the notion of epochs, the approach presented next improves system availability by efficiently using spare but paid resources in commonly available cloud environments.

Introduction and Motivation

ESP systems are naturally highly dynamic systems as they process data often originating from live data sources such as Twitter or Facebook where the streams have highly varying data rates that may change by several orders of magnitude within relative short periods of time. In order to cope with those peak loads and to prevent the unresponsiveness of the system, the systems are usually run at conservative utilization levels often as low as 50%. Although the cloud computing model enables customers to acquire resources quite easily, migration and rebalancing mechanisms are still not fast enough to accommodate sudden load spikes forcing the service providers to run their applications at low utilization levels. However, cloud users are nevertheless charged by full hours regardless of the actual resource consumption of their virtual machines.

Approach

In order to fully utilize all available resources a virtual machine offers, a *hybrid approach* is used for fault tolerance where a transition between *active replication* and *passive standby* based on the utilization of the system is made. In order to transition between the two states, a *priority scheduler* is used that prioritizes the processing of the primary and transparently pauses secondaries once resources are exhausted. Hence, the system

transparently transitions between active replication and passive standby where the secondary is paused until sufficient resources become available again. In order to keep the secondary up-to-date during high system utilization, the state of the primary is periodically checkpointed to the memory of the secondary which allows the secondary to prune enqueued but not yet processed events from queues. Using an interleaved partitioning scheme for the placement of primary and secondaries across the cluster, the overhead imposed on nodes during system recovery can furthermore be decreased.

Evaluation and Results

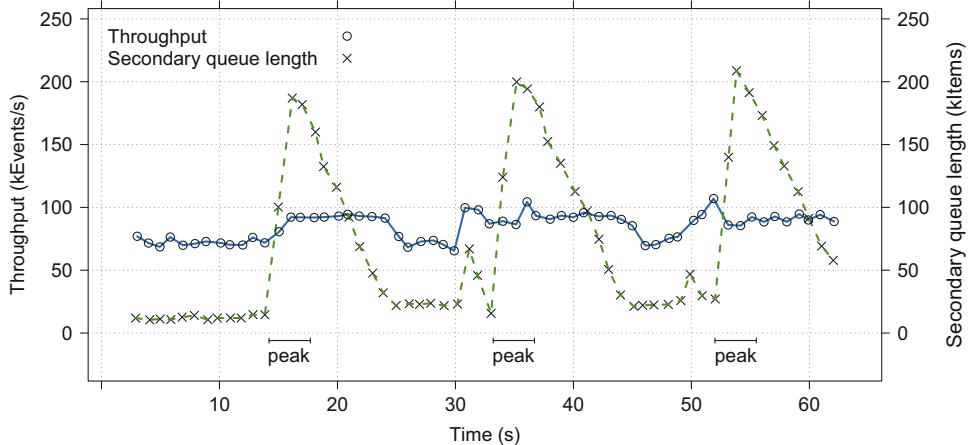
In the following experiment, the system behavior of the proposed solution has been investigated in the event of load peaks. To simulate spikes, a load generator to emit events at different rates for pre-defined periods of time was used. Figure 5 depicts the aggregated throughput for a single node and the status of the input queues of secondary slices on that node over time. In this experiment, the load generator introduced load spikes every 20 s for 2 s.

During a load peak, no events at the secondary slices on that node are being processed; hence queues grow quickly. Once the load decreases, secondaries resume the processing of events; hence, the amount of events in the queues of the secondary slices shrink. Note that the aggregated throughput of the node remains high until the shrinking process has been fully completed. During the spike, the aggregated throughput was higher due to the increase in load on the primary slices; after the spike, the throughput is higher due to the accumulated load on the secondary slices.

S

Adaptive and Low-Cost Fault Tolerance for Cloud Environments

In the previous section, an approach to utilize spare but already paid resources was presented that improves system availability by dynamically transitioning between active replication and passive standby during runtime. The following



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 5 Event throughput and queue length behavior of secondary slice queues with induced load spikes

section presents an approach that lowers the overall resource consumption by selecting the fault tolerance scheme at runtime that consumes the least amount of resources while still guaranteeing user-defined constraints such as *recovery time* and *recovery semantics*.

Introduction and Motivation

Fault tolerance in ESP systems can be carried out through various mechanism and replication schemes. For example, in *active replication*, redundant processing is used as a mechanism to carry out fault tolerance whereas in *passive replication*, a combination of periodic checkpointing and event logging is used in order to mitigate system crashes. Although active replication provides a quick recovery, it comes with a high price as it consumes almost twice of the resources, while passive replication consumes only little resources; however, it suffers from a long recovery time. Besides active and passive replication, there exist several more schemes to carry out fault tolerance such as *active and passive standby* where recovery time is traded by resource usage.

Choosing the right fault tolerance scheme is not trivial as all those schemes have different resource footprints and recovery times. Moreover, the footprint and recovery time for those schemes are not static as they strongly depend on system parameters that can greatly vary during the course

of processing. For example, the recovery time for passive replication strongly depends on the size of the checkpoint that must be read during a recovery. However, the size of a checkpoint strongly depends on the incoming data rate when considering a stateful sliding window operator.

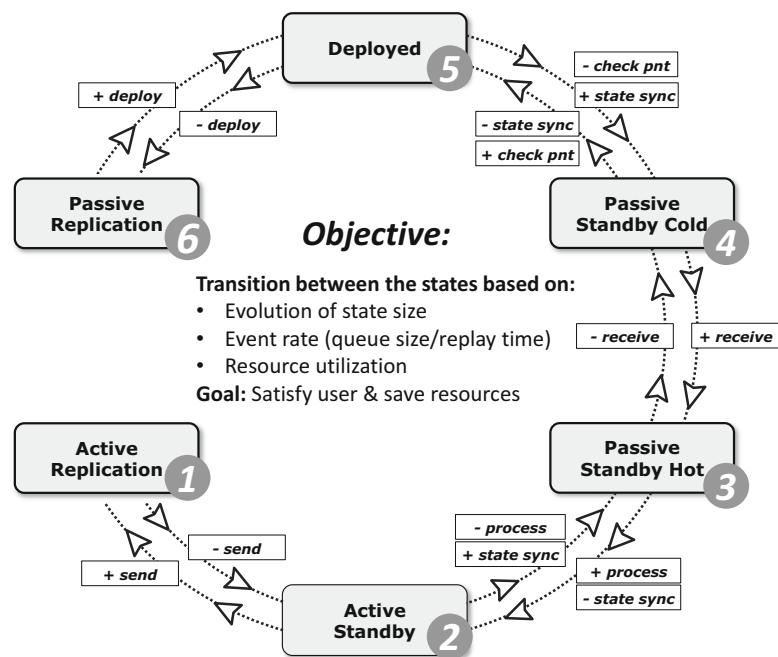
Approach

In order to free the user from the burden of choosing an appropriate fault tolerance scheme for his application, a *fault tolerance controller* that takes decisions on behalf of the user at runtime can be utilized. The controller takes into account user-provided constraints such as the desired *recovery time* and *recovery semantics*, i.e., precise or gap recovery.

Therefore, STREAMMINE3G was extended to support six different fault tolerance schemes the controller can choose from as shown in Fig. 6: ① *active replication*, ② *active standby*, ③ *passive standby hot* and ④ *cold*, ⑤ *deployed*, and ⑥ *passive replication*. The schemes have different characteristics with regard to resource consumption of CPU, memory, network bandwidth, the amount of nodes used, and recovery time. In order to choose the correct scheme, the controller uses an estimation-based approach where historically collected measurements are continuously

StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing,

Fig. 6 Fault tolerance schemes state transition wheel: active replication, active standby, passive standby hot and cold, deployed, and passive replication

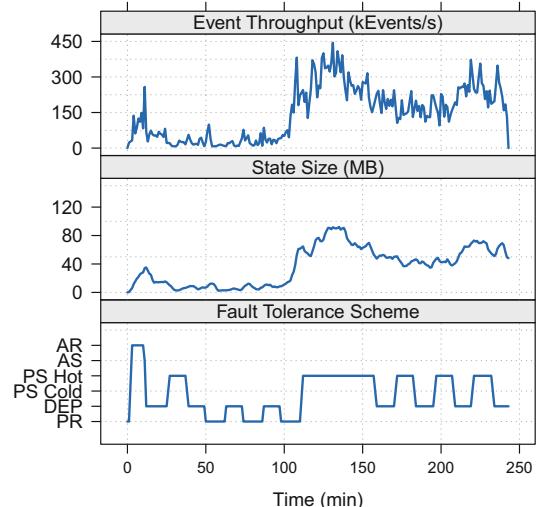


evaluated for an estimation of the expected recovery time for each of the available schemes.

Evaluation and Results

The approach has been evaluated with regard to the amount of resources that can be saved in comparison to a conservative use of full active replication. Figure 7 shows the runtime behavior of the system.

At the top graph, the throughput and how it varies over time are shown. Since the operator used for evaluation is a sliding window operator that accumulates events of the past 20 s, the size of the state follows the pattern of the throughput curve. At the bottom graphs, the chosen fault tolerance scheme is shown for each time slice. As shown in the plot, the system starts with active replication (AR), as it is the safe choice. Once enough measurements have been collected, the controller quickly switches to the deployed state replication scheme (DEP) as the state size and the throughput are quite low and, hence, recovery from disk and replay from upstream nodes can be easily accomplished within the user's specified recovery time threshold. However, as spikes occur which let the state and upstream queues



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 7 Throughput, state size, and fault tolerance scheme evolution over time using Twitter workload with a recovery threshold set to 5.5 s

grow, the controller switches between passive replication (PR) and deployed replication scheme (DEP). A cooldown time of 5 s prevents the system from oscillating due to sudden load spikes which are common in workloads originating

from live data sources such as Twitter streams. In summary, the controller chose a combination of passive replication and deployed during the first half of the experiment, whereas the second half was dominated by passive hot standby (PS Hot).

Conclusions

This chapter presented several approaches for lowering the overhead imposed by fault tolerance mechanisms in ESP systems. First, a brief overview of STREAMMINE3G was provided, the ESP system used to study and evaluate the presented approaches, followed by the first approach that allowed to reduce the overhead of event ordering required to recover applications in a precise manner and to ensure reproducibility through the use of *epochs* for *commutative* and *tumbling windowed operators* (Martin et al., 2011b). In order to apply this concept also for actively replicated operators, an extension to this approach was presented where the processing of epochs is delayed and by performing an *epoch-based deterministic merge*. In conjunction with a *lightweight consensus protocol*, latency as well as the propagation of non-determinism can be reduced and prevented, respectively.

Next, an approach to increase system availability by efficiently using spare but paid cloud resources (Martin et al., 2011a) was presented. The approach combines the two replication schemes *active replication* and *passive standby* where the system transparently switches between the two states using a *priority scheduler*. The evaluation showed that the system maintains responsiveness while still providing high availability through active replication at (almost) no cost.

As a last approach, a *fault tolerance controller* (Martin et al., 2015) was presented that selects an appropriate fault tolerance scheme on behalf of the user at runtime based on previously provided constraints such as recovery time and recovery semantics. The evaluation revealed that a considerable amount of resources can be saved

in comparison to the conservative use of active replication using this approach.

Cross-References

- [Apache Flink](#)
- [Elasticity](#)
- [Introduction to Stream Processing Algorithms](#)
- [Apache Kafka](#)
- [Apache Samza](#)
- [Stream Benchmarks](#)
- [Types of Stream Processing Algorithms](#)

References

- Apache s4 – distributed stream computing platform (2015). <https://incubator.apache.org/s4/>
- Apache samza – a distributed stream processing framework (2015). <http://samza.incubator.apache.org/>
- Apache storm – distributed and faulttolerant realtime computation (2015). <https://storm.incubator.apache.org/>
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
- Gu Y, Zhang Z, Ye F, Yang H, Kim M, Lei H, Liu Z (2009) An empirical study of high availability in stream processing systems. In: Proceedings of the 10th ACM/IFIP/USENIX international conference on middleware, Middleware '09, New York, pp 23:1–23:9. Springer, New York
- Hadoop mapreduce open source implementation (2015). <http://hadoop.apache.org/>
- Hunt P, Konar M, Junqueira FP, Reed B (2010) Zookeeper: wait-free coordination for Internet-scale systems. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC'10, Berkeley. USENIX Association, pp 1–14
- Hwang J-H, Balazinska M, Rasin A, Cetintemel U, Stonebraker M, Zdonik S (2005) High-availability algorithms for distributed stream processing. In: Proceedings of the 21st international conference on data engineering, ICDE '05, Washington, DC. IEEE Computer Society, pp 779–790
- Martin A, Fetzer C, Brito A (2011) Active replication at (almost) no cost. In: Proceedings of the 2011 IEEE 30th international symposium on reliable distributed systems, SRDS '11, Washington, DC. IEEE Computer Society, pp 21–30
- Martin A, Knauth T, Creutz S, Becker D, Weigert S, Fetzer C, Brito A (2011) Low-overhead fault tolerance for high-throughput data processing systems. In: Proceedings of the 2011 31st international conference on dis-

- tributed computing systems, ICDCS '11, Washington, DC. IEEE Computer Society, pp 689–699
- Martin A, Smaneoto T, Dietze T, Brito A, Fetzer C (2015) User-constraint and self-adaptive fault tolerance for event stream processing systems. In: Proceedings of The 45th annual IEEE/IFIP international conference on dependable systems and networks (DSN 2015), Los Alamitos. IEEE Computer Society
- Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: Proceedings of the 2010 IEEE international conference on data mining workshops, ICDMW '10, Washington, DC. IEEE Computer Society, pp 170–177
- Schneider FB (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv 22(4):299–319

Stream-Relational Queries

- ▶ [Continuous Queries](#)

StreamSQL Queries

- ▶ [Continuous Queries](#)

Structures for Big Data

- ▶ [Structures for Large Data Sets](#)

Structures for Large Data Sets

Peiquan Jin
 School of Computer Science and Technology,
 University of Science and Technology of China,
 Hefei, China

Synonyms

[Structures for big data](#); [Structures for massive data](#)

Definitions

Bloom filter (Bloom 1970): Bloom filter is a bit-vector data structure that provides a compact representation of a set of elements. It uses a group of hash functions to map each element in a data set $S = \{s_1, s_2, \dots, s_m\}$ into a bit-vector of n bits.

LSM tree (O’Neil et al. 1996): The LSM tree is a data structure designed to provide low-cost indexing for files experiencing a high rate of inserts and deletes. It cascades data over time from smaller, higher performing (but more expensive) stores to larger less performant (and less expensive) stores.

Skip list (Black 2014): Skip list is a randomized variant of an ordered linked list with additional, parallel lists. Parallel lists at higher levels skip geometrically more items. Searching begins at the highest level, to quickly get to the right part of the list, and then uses progressively lower level lists. A new item is added by randomly selecting a level, then inserting it in order in the lists for that and all lower levels. With enough levels, the time complexity of searching is $O(\log n)$.

Hash table (Cormen et al. 2009): Hash table is a dictionary in which keys are mapped to array positions by hash functions. Having the keys of more than one item map to the same position is called a collision. There are many collision resolution schemes, but they may be divided into open addressing, chaining, and keeping one special overflow area.

S

Overview

This subject is mainly toward data structures for large data sets, e.g., web pages, logs, and IoT (Internet of Things) sensing data. With the rapid development of big data applications such as web and IoT applications, people have to deal with massive data that cannot be efficiently processed and stored using traditional data structures.

This entry focuses on some typical structures that have been widely used for big data representation and organization. Differing from traditional structures such as B+-tree, the structures

discussed in this entry are specially designed for large data sets. After a brief description on the key ideas of the big-data-oriented structures, some examples of application are presented. And finally, this entry suggests a few future research directions in this field.

Key Data Structures

Large data sets introduce new challenges for the underlying structures for organizing data. First, the high velocity of big data calls for write-optimized data structures that can offer a high throughput for data insertions and deletions. Second, querying processing on large data sets costs more time, which seriously throttles read performance. Thus, most of structures for large data sets aim to improve the write and read performance on big data. However, many of them focus on finding a tradeoff between read optimization and write optimization, because many structures are not possible for optimizing both read performance and write performance. On the other hand, real applications usually exhibit an asymmetric property in read and write requests, i.e., some applications are write-intensive, while others are read-intensive. To this end, we can choose specifically optimized structures for different applications on large data sets.

In this section, we present some existing structures that are proposed for read/write optimization on large data sets.

Write-Optimized B+-tree

The B+-tree is a disk-based, paginated, dynamically updateable, balanced, and tree-like index structure (Liu and Özsü 2009). It supports point queries in $O(\log_p n)$ I/Os, where n is the number of records in the tree and p is the page capacity in number of records.

While the B+-tree provides efficient and stable performance for point queries, it has poor insertion performance. Each insertion in the B+-tree has to search from the root node to the appropriate leaf node, which costs $O(\log_p n)$ I/Os. In addition, it may incur iterated update of the B+-tree in order to keep the properties of the

tree such as balanced structure and approximately half filling-rate of each node.

Some write optimizations for the B+-tree are as follows (Bender and Kuszmaul 2013; Graefe 2004).

Insert in sequential order. The B+-tree is a couple of orders of magnitude faster at inserting data in sequential order compared to the random order. This is because once we insert a row into a leaf node, that leaf node is in memory, and since the next run of rows are destined for the same leaf, they can all be inserted cheaply.

This property of B+-tree leads many people to bend over backwards trying to keep insertions sequential. In many applications, this is not a natural or easy thing to do, and causes all sorts of problems elsewhere in the system.

Use a write buffer. This optimization stores up a bunch of insertions in a write buffer. When the buffer is full, we pick a leaf and write all the rows to the leaf.

This optimization works when we get to pick a bunch of stored rows that are going to the same leaf. When this happens, we see a speedup: you get to accomplish a bunch of work just for touching a leaf once.

The problem of this optimization is that we have to query the write buffer when answering queries, because the update-to-date rows may resist in the write buffer. However, as memory access is far fast than disk IOs, this approach can still have good read performance.

Write-optimized B+-trees received much attention in recent years, because of the popularity of the B+-tree in data management. It is a straightforward way to improve the B+-tree to meet the special needs of big data management. So far, write-optimized B+-trees are demonstrated advantageous for big data management, especially for streaming big data arriving at a high speed. The high throughput of insertion in write-optimized B+-trees makes it possible to meet the “velocity” challenge of big data.

The limitation of write-optimized B+-trees is that most of them sacrifice the read performance, which is not a good choice for read-intensive applications. As a result, it is necessary to devise

read/write-optimized B+-trees for big data management in future.

LSM-Tree

The Log-Structured Merge-Tree (or LSM-tree) (O’Neil et al. 1996) is a data structure proposed for highly inserted data, such as transactional log data. LSM-tree maintains data in two or more separate structures, each of which is optimized for its respective underlying storage medium. Data is synchronized in batches between the two structures efficiently.

One simple version of the LSM-tree is a two-level LSM-tree. A two-level LSM-tree comprises two tree-like structures, called C0 and C1. C0 is smaller and entirely resident in memory, while C1 is on disk. New records are first inserted into C0. If the insertion causes C0 to exceed a certain size threshold, a contiguous part of entries is removed from C0 and merged into C1 on disk. The LSM-tree has high update performance because movements from C0 to C1 are performed in batches, which implies that most writes to storage media are sequential writes.

Most implementations of LSM-tree used in practice employ multiple levels. Level 0 is kept in main memory and might be represented using a tree. The on-disk data is organized into sorted runs of data. Each run contains data sorted by the index key. A run can be represented on disk as a single file, or alternatively as a collection of files with nonoverlapping key ranges. To perform a query on a particular key to get its associated value, one must search in the Level 0 tree and each run.

The most significant advantage of the LSM-tree is its high performance of writing, because the writes from one level to its next level are always performed in sequential order. Thus, the LSM-tree is much suitable for big data applica-

tions with high-velocity data streams. The problem of the LSM-tree is its read performance. A read request in the LSM-tree may result in a few IOs to the levels which are maintained in disks.

Tree Structures Optimized for New Storage

Traditional structures were mainly proposed for magnetic disks. Disk IOs causing significant access latency become a bottleneck of big data storage and management. Recently, the advent of new storage such as flash memory and phase change memory introduces new opportunities for accelerating the performance of accessing big data. In this section, some typical tree structures proposed for the context of big data and new storage are described, including FD-tree, HybridB-tree, and Bloom-tree.

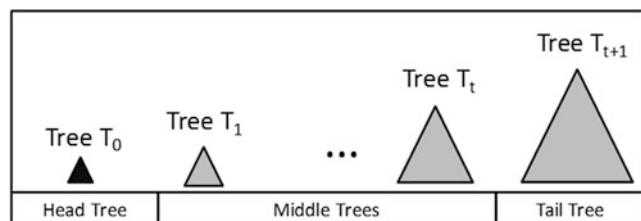
FD-Tree

The idea of FD-tree is to change random writes to an index into sequential ones (Li et al. 2010). FD-tree was originally proposed for flash memory. However, as random writes are also slower than sequential writes in hard disks, FD-tree can work on different types of storage.

FD-tree is a multi-layer tree index (as shown in Fig. 1), where the top layer is a two-level B+-tree called head tree, and each of the other layers consists of a sorted list. The updates to FD-tree are initially performed on the head tree, and then are gradually moved to the sorted lists at the lower levels. The key point of FD-tree is that each sorted list is organized as sequential pages that are contiguously stored in flash memory. Therefore, when flushing the updates to the index to flash memory, FD-tree only involves sequential writes to flash memory. As sequential writes to flash memory are more efficient than random writes, FD-tree can improve the overall

Structures for Large Data Sets, Fig. 1

Structure of the FD-tree



time performance by transforming random writes to sequential ones.

HybridB Tree

The HybridB tree (Jin et al. 2015) was designed for solid-state drives (SSD) and hard disks (HDD) based hybrid storage systems. It is an optimized structure of the B+-tree. In the HybridB tree, all the interior nodes are stored on SSD. Compared with leaf nodes, interior nodes in the HybridB tree are more possible to be read, because each read to a leaf node has to incur several reads to the interior nodes on the path from the root to the leaf node. On the other hand, all the updates, insertions, and deletions to the index are first focused on leaf nodes. Although there are possible updates to some interior nodes when updating a leaf node, these updates to interior nodes are much less than those to leaf nodes. As one major objective of the HybridB tree is to reduce random writes to SSD, it is more appropriate to put leaf nodes on HDD.

The structure of the HybridB tree is shown in Fig. 2. The leaf nodes in the tree are designed as *huge leaf* nodes (the yellow parts in Fig. 3), which are distributed among HDD and SSD. An interior node contains exactly one page, but a *huge leaf* node includes two or more pages. Each

page in a *huge leaf* node can be a *leaf-head* node, a *leaf-leaf* node, or a *leaf-log* node. The reason of introducing the *huge leaf* nodes in the HybridB tree is to reduce the splits/merges to the index, because these operations will incur many random writes to SSD.

The HybridB tree has the similar search cost with the B+-tree on SSD/HDD. However, owing to the use of the huge nodes, it has fewer random writes to SSD compared with the B+-tree on SSD/HDD. Generally, the HybridB tree can achieve better overall performance than the B+-tree, especially in a big data environment, because it can get more benefits over the B+-tree when the height of the tree increases.

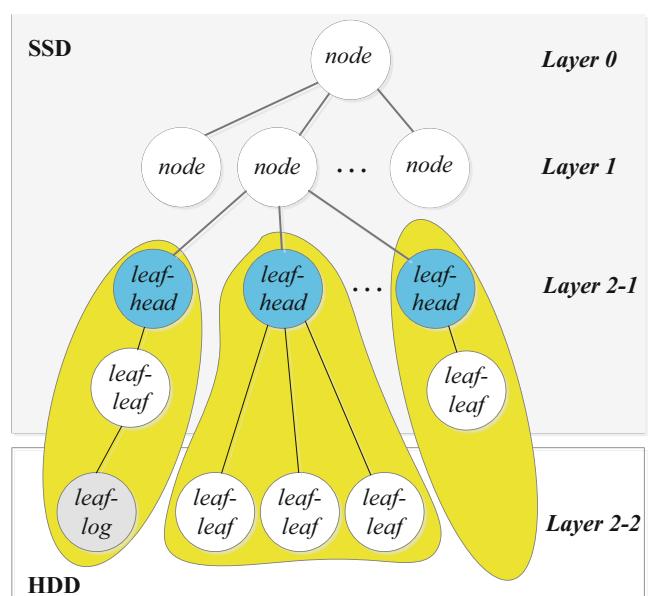
Bloom-Tree

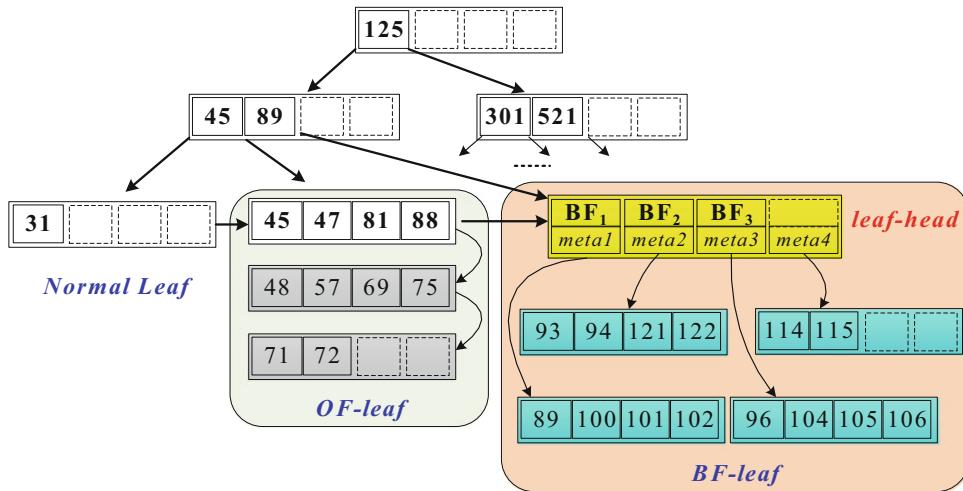
The Bloom-tree (Jin et al. 2016) is an efficient index structure proposed for large data sets on flash memory. It is a read/write-optimized tree structure of the B+-tree.

B+-tree has been demonstrated as an efficient tree index that has fast search performance on block-based storage such as hard disks. However, updates on B+-tree will incur a substantial number of page writes to disks in order to maintain its key features of balanced tree and half node-filling. As random writes are much slow on flash

Structures for Large Data Sets, Fig. 2

Structure of the HybridB tree



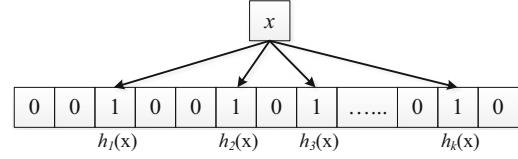


Structures for Large Data Sets, Fig. 3 Structure of the Bloom-tree

memory, many previous researches have shown that B+-tree will result in poor overall performance when directly used for flash memory (Roh et al. 2009).

In order to optimize B+-tree for flash memory, one solution is to cache the updates to the tree in a write buffer. Then, the cached updates can be merged and flushed into flash memory w.r.t. some optimal policy to reduce the overall writes on the tree index. However, even a write buffer is used for adapting B+-tree for flash memory, split operations on the tree will still lead to many page writes, as a split operation usually incurs propagating updates from a leaf node to the root of B+-tree. An intuitive way for this problem is to allow overflow pages for leaf nodes. Thus, newly inserted records can be put in overflow pages and splits will not be triggered by insertions.

So far, most flash-memory-oriented indices are designed based on the write buffer and/or the overflow page idea, aiming to reduce writes on the B+-tree. However, these solutions typically have to introduce many additional read operations. Meanwhile, modern flash disks show a close gap between their read and write speed, thus read operations on flash memory are becoming a critical factor affecting the overall time performance on flash memory. As a consequence, how to optimize flash-memory-aware indices by



Structures for Large Data Sets, Fig. 4 Element map to a Bloom filter

reducing both write and read costs is becoming an open challenge.

The Bloom-tree is able to control read costs while reducing writes to flash memory. In particular, it uses an update buffer and overflow pages to reduce random writes to flash memory and further exploits bloom filter to reduce the extra reads to the overflow nodes in the tree. With this mechanism, it constructs a read/write-optimized structure that can obtain better overall performance than previous flash-aware indices.

Figure 4 shows the structure of Bloom-tree. It improves the traditional B+-tree with two new designs. First, it introduces three kinds of leaf nodes, namely *Normal Leaf*, *Overflow Leaf (OF-leaf)*, and *Bloom-Filter Leaf (BF-leaf)*. Second, it proposes to construct bloom filters in *BF-leaf* nodes and use overflow pages in *OF-leaf* nodes.

A normal leaf node is the same as a leaf node on the traditional B+-tree, and it occupies exactly one page. An *OF-leaf* node contains overflow

pages. However, an *OF-leaf* node contains not more than three overflow pages in order to reduce read costs on *OF-leaf* nodes. If an *OF-leaf* node expands up to more than three pages, it will be transformed into a *BF-leaf* node, which can provide more efficient organization for overflow pages. A *BF-leaf* node is designed for organizing leaves with more than three overflow pages. As shown in Fig. 3, it contains several data pages and a *leaf-head* page maintaining the bloom filters and metadata about the data.

Hashing Structures for New Storage

The idea of hashing has emerged as a basic tool in data structures and algorithms. A wide range of algorithmic problems can be solved by using hash-based structures.

A hash table is a data structure that can map keys to values (Cormen et al. 2009). A hash table uses a hash function to map a set of keys to an array of buckets. By using hash tables, people can efficiently perform searching on large data sets. For example, if you need to find out in web logs the IP address with the highest access frequency to a specific website, a hash table can be used to partition all the possible IP addresses (2^{32}) into 1024 (or more) buckets. These buckets can be maintained in memory, and the IP address with the highest frequency can be easily computed.

The hash function for a hash table is expected to map each key to only one bucket. However, in many cases, the hash function may generate the same bucket number for several keys. This causes hash collisions in hash tables (Knuth 1998). There are two basic ways to resolve hash collisions. The first way is called *separate chaining*. In this method, each bucket is appended with a list of entries. If a few keys fall in the same bucket, they will be put into the list of the bucket. The separate chaining approach will introduce extra search costs for the list lookup. The second way is called *open addressing*. In this strategy, all entry keys are stored in the bucket itself. When a new key has to be inserted, the buckets are examined, starting with the hashed-to bucket and proceeding in some probe sequence, until an unoccupied bucket is found. When searching for an entry, the buckets are scanned in the same

sequence, until either the target record is found or an unused bucket is found, which indicates that there is no such key in the hash table. The name “*open addressing*” means that the address of a key is not determined by its hash value.

In recent years, there are some studies focusing on optimizing hashing structures for new storage such as flash memory. Flash memory has faster read speed than disks, but its random write speed is slower. Thus, hashing structures for flash memory mostly aim to reduce random writes to flash memory. MicroHash (Zeinalipour-Yazti et al. 2005) is an efficient external memory index structure for wireless sensor devices, which stores data on flash by time or value. MLDH (Multi Level Dynamic Hash index) (Yang et al. 2009) is a multi-layered hash index, and the capacity of each level in MLDH is twice as its upper level. Updates to MLDH are first buffered in an in-memory structure. When the memory structure is full, the memory data are merged with the hash index on flash memory and a new hash index is built. Wang et al. proposed a flash-based self-adaptive extendible hash index where each bucket occupies a block (erase unit) of flash memory (Wang and Wang 2010). A bucket consists of both data region and log region. The log region serves updates and deletes, so in-place updates to the data region can be delayed. In addition, a Split-or-Merge (SM) factor, which is dynamically adjusted according to the log/data ratio, is introduced to make the index self-adaptive. Hybrid Hash Index (Yoo et al. 2012) delays split operations which cause additional writes and erasure operations by using overflow buckets. Li et al. proposed a lazy-split hashing scheme to reduce writes at the expense of more reads (Li et al. 2008). They declared that the lazy-split hashing can adapt itself dynamically to different search ratios. Yang et al. proposed SAL-Hashing (Self-Adaptive Linear Hashing) (Yang et al. 2016) to reduce small random-writes to flash memory that are caused by index operations.

In-Memory Structures for Large Data Sets

In-memory data placement and processing has been regarded as a basic principle for ensuring fast read/write speed in big data management.

For this purpose, efficient in-memory structures are needed to be specially designed. This issue is especially important as the memory size is increasing with time. In this section, several structures proposed for in-memory data placement and accesses are discussed, including Bloom filter and Skip list.

Bloom Filter

Bloom filter is a space-efficient in-memory data structure to represent a set to which we can add elements and answer membership queries approximately (Bloom 1970). It has been extensively used as auxiliary data structures in database systems. Bloom filter is a bit-vector data structure that provides a compact representation of a set of elements. It maps each element in a data set $S = \{s_1, s_2, \dots, s_m\}$ into a bit-vector of n bits, which is denoted by $\text{BF}[1], \dots, \text{BF}[n]$ with a default value of 0. In this mapping procedure, we use a group of independent hash functions $\{h_1, h_2, \dots, h_k\}$. For a given element in S , each hash function maps it into a random number within the set of $\{1, 2, \dots, n\}$. If a hash function returns i , we set $\text{BF}[i]$ to 1. Figure 4 shows the mapping idea of Bloom filter.

For an element $x \in S$, the algorithm to compute its bloom filter includes two steps. First, it calculates the k values of hash functions $h_1(x), h_2(x), \dots, h_k(x)$. Second, it sets all bits $\text{BF}[h_i(x)]$ to 1.

For answering a membership query like “Is $y \in S?$,” the algorithm first calculates the k values of hash functions $h_1(y), h_2(y), \dots, h_k(y)$. Then, it checks all the Bloom filters of each element in S to see whether all the $\text{BF}[h_i(y)]$ in an existing Bloom filter are 1. If not, y is not a member of S . If all the $\text{BF}[h_i(y)]$ are 1, y may be within S . Due to the possible collisions of hash functions, there is a *false positive* rate when evaluating membership queries on Bloom filters. Given n, k , and m , the false positive probability of a Bloom filter can be computed by Eq. (1). Further, it is demonstrated that the false positive probability is minimalized when $k = 0.7 \cdot \frac{m}{n}$, which is approximately $0.6185^{\frac{m}{n}}$ (Bloom 1970).

$$f^{BF} = \left(1 - e^{-\frac{nk}{m}}\right)^k \quad (1)$$

Bloom filter is space efficient in representing elements. In addition, it is time efficient for inserting elements and answering membership queries. However, Bloom filter does not support deletions on elements. A revised version enhances Bloom filter with deletions (Bonomi et al. 2006).

Skip List

Skip list is a linked-list-like in-memory structure that allows fast search in memory (Pugh 1990). It consists of a base list holding the elements, together with a tower of lists maintaining a linked hierarchy of subsequences, each skipping over fewer elements.

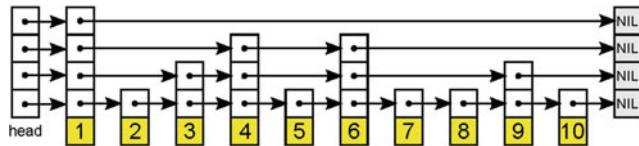
Skip list is a data structure that can be used in place of balanced trees. It uses probabilistic balancing rather than strictly enforced balancing and as a result the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees. Skip lists are built in layers, as shown in Fig. 5. The bottom layer is an ordinary ordered linked list. At a high level, a skip list is just a sorted, singly linked list with some “shortcuts” (additional pointers that allow traveling faster. An element in layer i appears in layer $i + 1$ with some fixed probability p (two commonly used values for p are $1/2$ or $1/4$). On average, each element appears in $1/(1-p)$ lists, and the tallest element (usually a special head element at the front of the skip list) in all the lists.

A search for a target element begins at the head element in the top list and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list.

Indeed, a skip list is really just a simulation of a binary search tree using multiple linked lists running in parallel. Of course, trying to maintain a perfectly balanced binary search tree using this simulation is still expensive, and it is still complicated. However, by viewing a skip list as

Structures for Large Data Sets, Fig. 5

Structure of skip lists



a single sorted linked list, where each node has a column of links to other nodes in the list, it is easy to see how randomly choosing a height for each new column could potentially produce the same good behavior as a balanced skip list.

The original design of the skip list was as a randomized data structure, much like randomized binary search trees. By eschewing a logarithmic bound guarantee in favor of a logarithmic bound with high probability, a skip list can enjoy the performance properties of a well-balanced binary search tree (on average) while avoiding many disadvantages of tree structures.

Compared with the balanced trees, the skip list has the following advantages:

1. It is relatively easy to design and implement the algorithm. The implementation of the skip list is direct and easier than the balanced trees.
2. It is efficient. Insertion and deletion do not need to be balanced again. Indeed, the skip list serves as an alternative to the binary search trees which become unbalanced after several insertion and deletion operations.
3. The memory requirements of skip lists are less than that used for balanced trees.

In the big data scope, the skip list is commonly employed as an efficient structure for organizing memory data. For example, it has been implemented in a couple of NoSQL database engines like LevelDB, Redis, and MemSQL as the primary memory structure.

Examples of Application

In many cases, hash tables are more efficient than trees or any other table structures. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and large sets.

Recent research by Google (Henzinger 2006; Das et al. 2007) showed that using deep learning approach can optimize hash functions by orders of magnitude in space usage savings and improve retrieval speed as well.

Hash tables may also be used as disk-based data structures and database indices. In multi-node database systems, hash tables are commonly used to distribute rows among nodes, reducing network traffic for hash joins (Karger et al. 1997).

Hashing is also commonly used in information retrieval. One early application in this field is identifying near-duplicate web pages in Altavista using min-wise hashing (Broder et al. 1998). Another application is HyperANF (Boldi et al. 2011), which was used to compute the distance distribution of the Facebook social network. In the field of machine learning, random mappings of data to lower-dimensional vectors that are easier to handle is of increasing importance for big data applications. This is because that machine learning algorithms often work with kernelized feature vectors with high dimensions. Designing randomized mappings that meet the criteria of machine learning applications has been an active research area in recent years (Wang et al. 2016).

A key application of Bloom filters is in the field of content delivery networks, which deploys web caches around the world to cache and serve web content to users with greater performance and reliability. Bloom filters are used to efficiently determine which web objects to store in web caches (Maggs and Sitarasan 2015). To prevent caching one-hit-wonders (accessed by users only once and never again), a Bloom filter is used to keep track of all URLs that are accessed by users. A web object is cached only when it has been accessed at least once before. With this mechanism, one can significantly reduce the disk write workload, since one-hit-wonders are never written to the disk cache. Further, filtering out the

one-hit-wonders also saves cache space on disk, increasing the cache hit rates.

Bloom filters can also be organized as distributed data structures to perform fully decentralized computations of aggregate functions (Pournaras et al. 2013). Decentralized aggregation makes collective measurements locally available in every node of a distributed network without involving a centralized computational entity for this purpose.

The LSM-tree is now used in many database products such as Bigtable, HBase, LevelDB, MongoDB, SQLite, RocksDB, WiredTiger, Apache Cassandra, and InfluxDB. The LSM-tree is especially suitable for write-intensive workloads. Certainly, LSM implementations such as LevelDB and Cassandra regularly provide better write performance than single-tree based approaches. Yahoo! developed a system called PNUTS, which combines the LSM-tree with the B-trees and demonstrates better performance (Cooper et al. 2008).

It is also worth considering the hardware being used. Some expensive solid state disks, like FusionIO, have better random write performance. This suits update-in-place approaches. Cheaper SSDs are better suited to the LSM-tree, because it can avoid small random writes to SSDs.

Skip list has been implemented in many systems such as LevelDB, MemSQL, and Redis. Specially, Redis adds a backward pointer for each skip list node to traverse reversely. Also there is a span variable in level entry to record how many nodes must be crossed when reaching to next node. Actually, when traverse list, we can accumulate span to get the rank of a node in sorted set. In LevelDB, skip lists are used to organize the MemTable in memory. In MemSQL, skip lists are used as the prime indexing structure for databases.

Future Directions for Research

Data structures are one of the most critical parts for big data representation, processing, and storage. In addition to the structures explained

in this article, there are some future research directions in this field.

Firstly, the existing tree structures like LSM-tree and HybridB-Tree are mainly toward write-intensive workloads. Although it is a major objective to reduce random writes to new storage such as flash memory, it is valuable to devise read/write-friendly structures in the future.

Secondly, traditional database researches suppose that data are stored in external disks. However, big data applications call for in-memory data processing and storage. Thus, how to make data structures suit for in-memory data management is an important issue in the big data era.

Finally, recently nonvolatile memory (NVM) has received much attention from both academia and industries (Li et al. 2016; Chen et al. 2014). This will lead to reformation of memory architecture. Therefore, developing NVM-oriented structures is a potential research direction in the future.

Acknowledgments

We would like to thank University of Science and Technology of China for providing the environment where the study described in this entry was completed. The work involved in this entry is partially supported by the National Science Foundation of China (No. 61672479).

Cross-References

- ▶ [Big Data Indexing](#)
- ▶ [Search and Query Accelerators](#)

References

- Bender M, Kuszmaul B (2013) Data structures and algorithms for big databases. In: 7th extremely large databases conference, Workshop, and Tutorials (XLDB), Stanford University, California
- Black P (2014) Skip list. In: Pieterse V, Black P (eds) Dictionary of algorithms and data structures. <https://www.nist.gov/dads/HTML/skiplist.html>
- Bloom B (1970) Space/time trade-offs in hash coding with allowable errors. Commun ACM 13(7):422–426

- Boldi P, Rosa M, Vigna S (2011) HyperANF: approximating the neighbourhood function of very large graphs on a budget. In: Srinivasan S et al (eds) Proceedings of the 20th international conference on World Wide Web, March 2011, Hyderabad/India, p 625–634
- Bonomi F, Mitzenmacher M, Panigrahy R, Singh S, Varghese G (2006) An improved construction for counting Bloom filters. In: Azar Y, Erlebach T (eds) Algorithms – ESA 2006, the 14th annual european symposium on algorithms, September 2006, LNCS 4168, Zurich, Switzerland, p 684–695
- Broder A, Charikar M, Frieze A, Mitzenmacher M (1998) Min-wise independent permutations. In: Vitter J (eds) Proceedings of the thirtieth annual ACM symposium on the theory of computing, May 1998, Dallas, Texas, p 327–336
- Chen K, Jin P, Yue L (2014) A novel page replacement algorithm for the hybrid memory architecture involving PCM and DRAM. In: Hsu C et al (eds) Proceedings of the 11th IFIP WG 10.3 international conference on network and parallel computing, September 2014, Ilan, Taiwan, p 108–119
- Cooper B, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen H, Puz N, Weaver D, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. Proc VLDB Endowment 1(2):1277–1288
- Cormen T, Leiserson C, Rivest R, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Boston, pp 253–280
- Das A, Datar M, Garg A, Rajaram S (2007) Google news personalization: scalable online collaborative filtering. In: Williamson C et al (eds) Proceedings of the 16th international conference on World Wide Web, May 2007, Banff, Alberta, p 271–280
- Graefe G (2004) Write-Optimized B-Trees. In: Nascentino M, Özsu M, Kossmann D, et al. (eds) Proceedings of the thirtieth international conference on very large data bases, Toronto, Canada, p 672–683
- Henzinger M (2006) Finding near-duplicate web pages: a large-scale evaluation of algorithms, In: Efthimiadis E et al (eds) Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval, August 2006, Seattle, Washington, p 284–291
- Jin P, Yang P, Yue L (2015) Optimizing B+-tree for hybrid storage systems. Distrib Parallel Databases 33(3): 449–475
- Jin P, Yang C, Jensen C, Yang P, Yue L (2016) Read/write-optimized tree indexing for solid-state drives. VLDB J 25(5):695–717
- Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Leighton F et al (eds) Proceedings of the twenty-ninth Annual ACM symposium on the theory of computing, May 1997, El Paso, Texas, p 654–663
- Knuth D (1998) The art of computer programming. 3: sorting and searching, 2nd edn. Addison-Wesley, New York, pp 513–558
- Li X, Da Z, Meng X (2008) A new dynamic hash index for flash-based storage. In: Jia Y et al (eds) Proceedings of the ninth international conference on web-age information management, July 2008, Zhangjiajie, China, p 93–98
- Li Y, He B, Yang J, Luo Q, Yi K (2010) Tree indexing on solid state drives. Proc VLDB Endowment 3(1): 1195–1206
- Li L, Jin P, Yang C, Wan S, Yue L (2016) XB+-tree: a novel index for PCM/DRAM-based hybrid memory. In: Cheema M et al (eds) Databases theory and applications – proceedings of the 27th Australasian database conference, September 2016, LNCS 9877, Sydney, Australia, p 357–368
- Liu L, Özsu M (2009) Encyclopedia of database systems. Springer, New York
- Maggs B, Sitaraman R (2015) Algorithmic nuggets in content delivery. SIGCOMM Comput Commun Rev 45(3):52–66
- O'Neil P, Cheng E, Gawlick D, O'Neil E (1996) The log-structured merge-tree (LSM-tree). Acta Informatica 33(4):351–385
- Pournaras E, Warnier M, Brazier F (2013) A generic and adaptive aggregation service for large-scale decentralized networks. Complex Adapt Syst Model 1:19
- Pugh W (1990) Skip lists: a probabilistic alternative to balanced trees. Commun ACM 33(6):668
- Roh H, Kim W, Kim S, Park S (2009) A B-tree index extension to enhance response time and the life cycle of flash memory. Inf Sci 179(18): 3136–3161
- Wang L, Wang H (2010) A new self-adaptive extendible hash index for flash-based DBMS. In: Hao Y et al (eds) Proceedings of the 2010 IEEE international conference on information and automation, June 2010, Haerbin, China, p 2519–2524
- Wang J, Liu W, Kumar S, Chang S (2016) Learning to hash for indexing big data – a survey. Proc IEEE 104(1):34–57
- Yang C, Lee K, Kim M, Lee Y (2009) An efficient dynamic hash index structure for NAND flash memory. IEICE Trans Fundam Electron Commun Comput Sci 92(7):1716–1719
- Yang C, Jin P, Yue L, Zhang D (2016) Self-adaptive linear hashing for solid state drives. In: Hsu M et al (eds) Proceedings of the 32nd IEEE international conference on data engineering, May 2016, Helsinki, Finland, p 433–444
- Yoo M, Kim B, Lee D (2012). Hybrid hash index for NAND flash memory-based storage systems. In: Lee S et al (eds) Proceedings of the 6th international conference on ubiquitous information management and communication, February 2012, Kuala Lumpur, Malaysia, p 55:1–55:5
- Zeinalipour-Yazti D, Lin S, Kalogeraki V, Gunopulos D, Najjar W (2005) MicroHash: an efficient index structure for flash-based sensor devices. In: Gibson G (eds) Proceedings of the FAST '05 conference on file and storage technologies, December 2005, San Francisco, California, p 1–14

Structures for Massive Data

- ▶ Structures for Large Data Sets

Survey

- ▶ Big Data in Computer Network Monitoring

SUT

- ▶ System Under Test

SWAG

- ▶ Sliding-Window Aggregation Algorithms

System Under Test

Francois Raab
InfoSizing, Manitou Springs, CO, USA

Synonyms

Measured system; SUT; Target environment; Test object; Tested environment

Definitions

A System Under Test (SUT) is a complete system that comprises hardware, software, and connectivity components; that is the object or target of a performance measurement test or benchmark.

Historical Background

While the term System Under Test is somewhat generic in nature, it was formalized in the early 1990s by industry consortiums with the mission of defining industry standard performance benchmarks, such as the Transaction Processing Performance Council (TPC – www.tpc.org), the Standard Performance Evaluation Corporation (SPEC – www.spec.org), and the Storage Performance Council (SPC – www.storageperformance.org).

Foundations

The purpose of a test or benchmark is to determine how a target environment behaves under a controlled load or in response to specific requests. As such, the definition of the System Under Test plays a key role in a benchmark specification. The SUT's definition discusses the inclusion or exclusion of hardware, software, and connectivity components. It also defines the measurement boundaries. For benchmarks that mandate some pricing metrics, the SUT defines which components must be included in this pricing.

Hardware Components

The definition of the SUT includes a number of hardware components. These components can be specific or generic. Specific components can be chosen to define a fixed set of hardware on which to measure different software variations. By keeping the hardware as a constant while varying the software, the outcome of the test can be fully attributed to changes in the software environment. Alternatively, the hardware components can be defined in generic terms, giving some latitude in their selection. This latitude can be limited to some portion of the SUT or can be applied to all hardware components in the SUT. For example, the SUT's definition may require the configuration of a single server node with specific attached storage and a specific number of processor sockets, while giving latitude in the choice of processor types. Such a SUT definition would focus the results of the test of the

processor's capabilities. Alternatively, the SUT's definition may give full latitude in the selection of the server's internal architecture, system topology (single-node, cluster, client–server, etc.), and storage subsystem; giving the test an opportunity to compare widely diverging solutions to a common workload requirement, or analyzing non-hardware criteria, such as price, value (price/performance), scalability, or portability.

Software Components

The definition of the SUT generally includes a number of software components or a software stack. Few tests solely focused on a hardware component (e.g., a simple storage device) and do not require any software as part of the SUT. But, in most cases, one or more software components are needed to assemble the SUT. There is a degree to the role that software can play in the SUT. Software can be part of the SUT to simply tie together the hardware components targeted by the test. In the case of system testing, software components are just as important as hardware components for the tested system, literally a System Under Test. And for tests that directly target software, rather than hardware, the software components are the object of the test and the hardware components merely provide a platform for the software to execute.

Connectivity Components

The definition of the SUT may include a number of connectivity components. These are used to allow the hardware components to communicate. Such connectivity might be in the form of a general-purpose network (e.g., Ethernet over Fibre Channel) or in the form of a proprietary interconnect (e.g., AMD's HyperTransport bus). Many test environments use some connectivity components to provide a communication channel between the SUT and the test harness. These communication channels may or may not be included as part of the SUT. One criteria to include such communication channel in the SUT is whether they are also used for communication between other components inside the SUT.

Measurement Boundary

The boundary between a test driver (also called measurement harness) and the target of the measurement (i.e., the SUT) is the point where the driver applies a stimulus to the SUT and where the driver measures the SUT's response to that stimulus. This measurement boundary is also where the test driver ends and the SUT starts. In some cases, the measurement boundary is virtual in that the driver uses resources from components that are part of the SUT. In other cases, the boundary is physical in that no components, other than a communication channel, are shared between the driver and the SUT.

Pricing Metrics

When benchmarks results are used for competitive analysis, it may be relevant to include some pricing metrics. The purpose of such metrics is to provide information about the cost associated with the performance capabilities of the SUT. These metrics are generally expressed in terms of straight total price, in terms of ratio between total price and measured performance, or both. For example, the TPC-A metrics include the total cost of ownership of the SUT and the associated cost per unit of performance, expressed in cost/tps (transaction per second). The definition of the SUT is a key to defining which components are included in these cost calculations.

Key Applications

First Formal Use

The first formal use of the term SUT was found as part of the initial release of TPC Benchmark A (TPC-A) (Huppler 2009), published by the TPC in 1990. This online transaction processing (OLTP) benchmark provided an industry-sanctioned specification of the loosely defined and widely used DebitCredit (Anon 1985) workload. The TPC-A specification defined the term SUT as including the following components:

- *One or more processing units (e.g., hosts, front-ends, and workstations.), which will run the transactions [. . .].*

- *The hardware and software components of all networks required to connect and support the SUT components.*
- *Data storage media sufficient to satisfy both the scaling rules [...] and the ACID properties [...].*
- *The host system(s) including hardware and software supporting the database employed in the benchmark.*

Generic Terminology

Following in the footsteps of TPC-A, subsequent benchmark specifications released by industry standard consortia have been using the term SUT, or one of its synonyms, to identify the performance measurement target specific to each benchmark. The term SUT, by itself, does not describe the tested environment. Instead, the term is used as a container to encapsulate the set of hardware, software, and connectivity components that are being targeted for measurement. As such, each benchmark specification includes its own formal definition of the term SUT.

In some instances, the term used to name the target environment of a test is synonymous with System Under Test. For example, SPEC does not consistently use the term SUT in its benchmark specifications. However, the consortium uses the term SUT to name the set of configuration components that can be chosen as filters for queries against its database of benchmark publications.

Benchmark Adoption and Longevity

The SUT's definition is a key factor in a benchmark's adoption and longevity. According to (Huppler 2009), all “good” benchmarks share the same primary characteristics of being *relevant, repeatable, fair, verifiable, and economical*. A careful definition of the SUT will contribute to improving each one of these characteristics.

- Relevant: The SUT's primary components are those for which test results are in demand.
- Repeatable: The SUT's definition has tight requirements for components that impact the test results, and gives flexibility for components that are result neutral.
- Fair: Components of the SUT are limited to those that can be compared on an even playing field.
- Verifiable: The availability of the components used and their configuration is such that others can easily recreate the SUT to repeat the test.
- Economical: The cost of assembling the SUT does not create a prohibitive limit on who can reasonably execute the test.

Cross-References

- ▶ [Big Data Benchmark](#)
- ▶ [Benchmark Harness](#)
- ▶ [Big Data Architectures](#)
- ▶ [Cloud Big Data Benchmarks](#)
- ▶ [Component Benchmark](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [Energy Benchmarking](#)
- ▶ [Metrics for Big Data Benchmarks](#)
- ▶ [Microbenchmark](#)
- ▶ [TPC](#)

S

References

- Anon et al (1985) A measure of transaction processing power. Datamation 31:112–118
Huppler K (2009) The art of building a good benchmark. In: Nambiar R, Poess M (eds) Performance evaluation and benchmarking, vol 5895. Springer, Berlin/Heidelberg, pp 18–30. https://link.springer.com/chapter/10.1007/978-3-642-10424-4_3

T

Table Lookup

► [Tabular Computation](#)

Tabular Computation

Behrooz Parhami
Department of Electrical and Computer
Engineering, University of California, Santa
Barbara, CA, USA

Synonyms

[Approximate computation](#); [Iterative refinement](#);
[Table lookup](#)

Definitions

Big data necessitates the use of very large memories that can bring about other uses of such units, say, for storing precomputed values of functions of interest, to improve speed and energy efficiency.

Overview

Until the 1970s, when compact and affordable digital scientific calculators became available, we relied on pre-calculated tables of important

functions that were published in book form (e.g., Zwilfinger 2011). For example, base-10 logarithm of values in [1, 10], at increments of 0.01, might have been given in a 900-entry table, allowing direct readout of values if low precision was acceptable or use of linear interpolation to obtain greater precision. To compute $\log 35.419$, say, one would note that it is $1 + \log 3.5419 = 1 + \log 3.54 + \varepsilon$, where $\log 3.54$ is read out from the said table and ε is derived based on the small residual 0.0019 using some sort of approximation or interpolation. Once everyone became equipped with a sophisticated calculator and, later, with a personal computer, the use of tables fell out of favor. Table-based computation returned in at least two different forms in the 1990s. One was to speed up normal, circuit-based computations by providing an initial estimate that would then be refined. The other was to reduce complexity and power consumption.

Implications of High Volume for Big Data

Studies at IBM and Cisco Systems show that as of 2012, we generated 2.5 exabytes of data per day (Cisco Systems 2017; Jacobson 2013) [1 exabyte = 1 quintillion or 10^{18} bytes = 1 gigagigabyte], and this is set to explode to 40 yottabytes per day [1 yottabyte = 1 septillion or 10^{24} bytes = 1 teraterabyte] by 2020. Multiply the latter number by 365 and then by the number of years, and the extent of data becomes even

more mind-boggling. Organizing, managing, and updating such large volumes of data will be quite challenging.

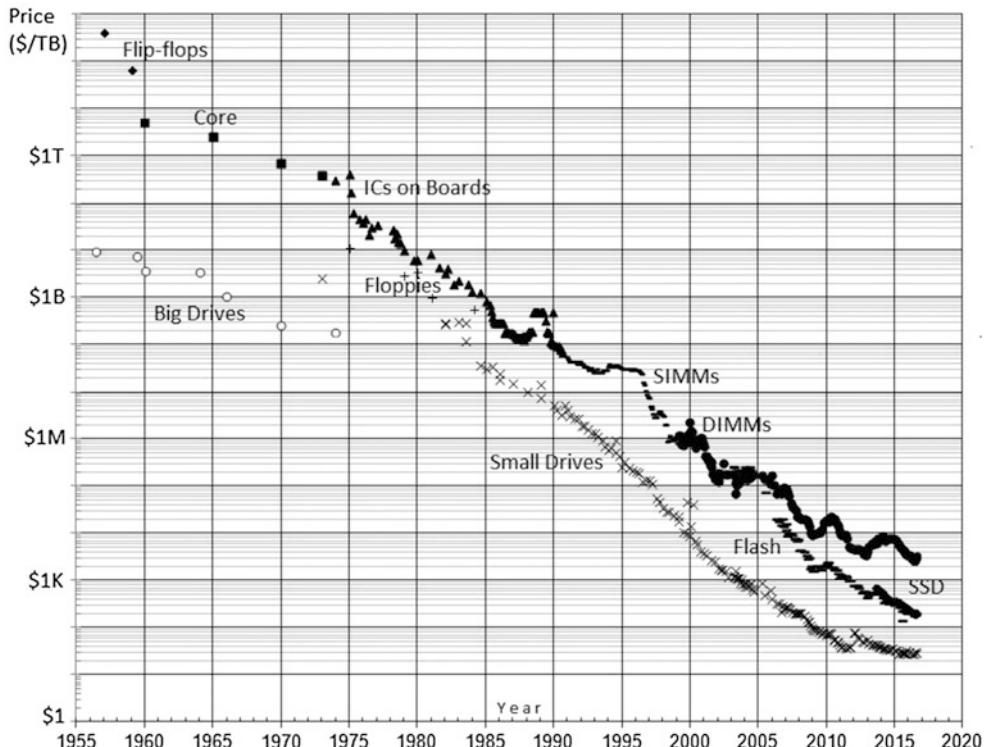
A direct consequence of high data volume is high computational load, as confirmed by a simple back-of-the-envelope calculation. Processing 1 petabyte of data, at around 100 clock cycles or instructions per byte, means a processing time of 10^{17} clock cycles. With an optimistic clock rate of 10 GHz (10^{10}), we will need a processing time of 10^7 s (~ 4 months). This is already impractically long, and it becomes even more so for data volumes much larger than a petabyte. It follows that replacing many cycles of computation with a single table lookup will offer immense performance benefits.

Simultaneously with the exponential growth of data production rate (Chen and Zhang 2014; Hilbert and Gomez 2011), we have been experiencing an exponential reduction in the cost of computer memory and storage, as depicted in

Fig. 1 (McCallum 2017). The reduced cost of memory and storage, combined with more advanced data compression schemes (Storer 1988), renders big-data applications feasible while also enabling new categories of applications, which would not even be contemplated in the absence of inexpensive storage. One such area is increased reliance on large-scale tables for performing or facilitating computation.

Why Table-Based Computation?

Function evaluation methods are studied in the field of computer arithmetic (Parhami 2010), dealing with algorithms and hardware designs for performing computations within arithmetic/logic units found as components of CPUs in general-purpose computer architectures (Parhami 2005), as well as within more specialized computational structures, such as graphic processing units or GPUs (Owens et al. 2008).



Tabular Computation, Fig. 1 Exponentially declining cost of computer memory and storage (McCallum 2017)

Computing any function requires time and other resources, such as energy. If a particular function value, say, $f(a)$, is needed many times in the course of different computations or the same computation performed on different inputs, it makes sense to store the computed value and use it any time $f(a)$ is needed, without having to recompute. Storage can be accomplished via conventional tables that are accessed by operand values used as index into the table or may entail some form of cache structure (Smith 1982) that is consulted before triggering the requisite calculations in the event of a cache miss (Gabbay and Mendelson 1998).

One way to reduce the required table size is to subject the operands to some preprocessing or to allow some post-processing of the value(s) read out from the table(s). This kind of indirect table lookup provides a range of trade-offs between pure table lookup and pure computational or circuit-based approach. For example, the two-operand multiplication operation $p(a, b) = ab$ can be performed via two accesses to a smaller squaring table, using the formula $ab = [(a + b)^2 - (a - b)^2]/4$, whose use entails two additions in the preprocessing phase and one addition along with a 2-bit right shift in the post-processing stage. Additional optimizations can be applied to this method (Vinnakota 1995).

Reading a stored value requires less energy than recomputing it, and reading may also be faster, particularly for a computationally complex function. Because table size grows exponentially with the number of bits in the input operand(s), the scheme is particularly efficient for low-precision data, although continual increase in size and reduction in cost of memory is expanding the method's applicability.

Low-precision computation, as embodied in the rapidly expanding field of approximate computing (Mittal 2016), is an increasingly important part of the workload in modern computing. One advantage of table-based approximate computing is that the exact error for each table entry is knowable, whereas in a circuit-based approach, often a bound on the error is the best we can provide.

Lookup Table Implementation Options

The conceptually simplest method for implementing a lookup table is to allocate 2^u words, each v bits wide, if there are u input operand(s) bits and the result is to be v bits wide. The table can then be filled with precomputed function values, essentially resulting in an electronic version of the mathematical tables of yore found in pages of handbooks.

This is rarely practical or efficient. One optimization is to allocate space for all table entries but only fill them when the values become needed for the first time. In this way, the table will act as a sort of cache memory which will generate a “miss” upon first access to a particular entry. When a miss is encountered, the requisite computation is performed to derive the value, which is then forwarded to both the computation that led to the miss and to the corresponding table entry for possible future use.

A second optimization is to look for simple transformations in the input operands that can be performed efficiently in terms of time and energy while leading to substantial savings in table size. If a two-operand function is symmetric, for example, the table size can be cut in half by determining which operand is larger and ordering the operands. Such optimizations are function-dependent and provide a wide range of trade-offs between computation time and lookup table size.

A third optimization is to divide the domain of interest into a number of subranges and implement a separate lookup scheme for each subrange. For subranges where the function variations are less pronounced, smaller tables will do, whereas subranges with significant variations necessitate larger lookup tables. Related to this last optimization is segmenting the domain of interest in a nonuniform manner, using a special mechanism to map input values into corresponding subranges and thus associated lookup tables (Lee et al. 2003).

An attractive alternative to parallel-access tables, with original or transformed operand bits presented in parallel to form an index into the

table, is a combination of bit-serial computation with table lookup. The scheme, known as distributed arithmetic (White 1989), allows certain computations to be implemented through table lookup, despite the large number of input values involved. The speed of distributed arithmetic can be respectable and the implementation cost modest if the computation is pipelined at a high clock rate. It is also possible to do intermediate schemes where the computation is not fully serial, but nibble- or byte-serial.

Table-based methods can be implemented in a variety of ways and new schemes continue to emerge in light of ongoing research. To keep our discussion manageable, we focus on the basics of two general methods, the interpolating memory scheme and multipartite table method, in the following two sections. More details on these methods, including additional references, as well as other pure and hybrid methods can be found in books, periodicals, and conferences on computer arithmetic (e.g., Parhami 2010).

Interpolating Memory

One can combine the lookup process for reading out approximate function values with the interpolation scheme needed for greater precision into a single-hardware unit, dubbed interpolating memory (Noetzel 1989).

Consider the computation of $f(a)$, where the parameter a falls between x_i and x_{i+1} , two consecutive lookup table indices, that is, $x_i < a < x_{i+1}$. Then, using linear interpolation, the value of $f(a)$ can be approximated as

$$f(a) = f(x_i) + (a - x_i) \times f'(x_i)$$

where $f'(x_i)$ is the derivative or slope of the function at x_i . More generally, the values $A_i = f(x_i)$ and $B_i = f'(x_i)$, or a value that better approximates intermediate values between $f(x_i)$ and $f(x_{i+1})$, are stored, with a post-processing circuit computing $f(a) = A_i + (a - x_i) \times B_i$.

Practically, x_i corresponds to several leading bits of a and $a - x_i$ is represented by the remaining

bits of a . This scheme simplifies the implementation and reduces its complexity. Let a_u be defined by a few upper bits of a , and let a_r be the value represented by the remaining bits of a , so that a_u and a_r collectively span all the bits of a . Then, $f(a) = f(a_u) + s(a_u) \times a_r$.

One can increase the computational precision by using second- or third-degree interpolation, which, for the same precision, would need smaller tables (Noetzel 1989) but requires more external circuitry, providing a spectrum of implementation options. For example, with second-degree interpolation, we have $f(a) = f(a_u) + s(a_u) \times a_r + t(a_u) \times a_r^2$. With the latter formula, a squarer and two multipliers are needed for highest-speed implementation. Alternatively, one can use a single multiplier to perform the operation $a_r \times a_r$, and the other two multiplications sequentially. If the function must be evaluated for several different x values, the computation can be pipelined, so that the maximum processing rate is achieved while using only a single multiplier and one adder.

Bipartite and Multipartite Tables

Interpolating memory, discussed in the preceding section, requires the use of a multiplier to achieve the greater precision afforded by linear interpolation. Multiplication is time-, cost-, and energy-intensive compared with addition, so table lookup schemes that avoid multiplication would be desirable if they offer adequate precision using only addition as pre- and post-processing operations. Schemes in this category are known as multiplier-less methods (e.g., Gustafsson and Johanson 2006).

Bipartite tables (Das Sarma and Matula 1995) offer one such scheme. Consider a k -bit operand divided into a few of its most-significant bits, x_u , a few middle bits, x_m , and the rest of the bits x_r , collectively spanning the entire width of the operand x . We can approximate $f(x)$ as the sum $g(x_u, x_m) + h(x_u, x_r)$. Essentially, the decomposition into x_u , x_m , and x_r creates intervals corresponding to different values of x_u

and subintervals corresponding to different values of x_m . Function values are stored for each subinterval in the table $g(x_u, x_m)$. The ingenuity of the method is that instead of storing slopes for the various subintervals, as in interpolating memory, a common slope for each interval is stored, allowing the multiplication of the slope $s(x_u)$ and the displacement x_r to be performed by the second lookup table which yields the value $h(x_u, x_r)$. Selection of the number of bits in x_u and x_m offers various tradeoffs in precision and table size, allowing the designer to choose an optimal scheme, given application requirements.

The basic ideas discussed above have been extended and refined in a variety of ways. Bipartite tables can be optimized by taking advantage of symmetry (Schulte and Stine 1999; Stine and Schulte 1999). Extension to tripartite (Muller 1999) and multipartite tables (De Dinechin and Tisserand 2005; Kornerup and Matula 2005) has also been attempted.

Iterative Refinement Methods

Table lookup can be used in conjunction with iterative refinement methods to obtain results of higher precision that are impossible or cost-ineffective to derive via pure table lookup. The iterative refinement process essentially replaces an interpolation scheme that would be much more complex, if the same precision were to be achieved. We present just one example, but there are many more.

Consider finding the square root of z , starting with an initial approximation $x(0)$. The following iterative scheme yields an approximation $x(j+1)$ with maximum error 2^{-2k} from a previous approximation $x(j)$ with maximum error 2^{-k} :

$$x(j+1) = \frac{1}{2}(x(j) + z/x(j))$$

The scheme's quadratic convergence ensures that a small number of iterations will suffice. For example, if an initial table entry provides the square root of z with 16 bits of precision, one iteration produces a 32-bit result and two iterations will yield a 64-bit result. The method just discussed has many variations and alternatives,

including versions that are division-free (Cray Research 1989), given that division is a slower and more complicated operation compared with multiplication.

Future Directions

After decades of being restricted to low-precision arithmetic, table lookup is emerging as a feasible method of attack to higher-precision computation, aided by the lower cost and higher density of memory devices as well as improved algorithms for reducing the required table size via advanced pre- and post-processing schemes.

In addition to general techniques discussed in the preceding sections, a large number of special algorithms and associated tweaks can be used to make implementations more efficient or better-suited to specific application domains. Methods such as multilevel table lookup (Parhami 1997); bit-level, application-specific optimization of lookup tables (Parhami and Hung 1994); and accurate error analysis (Tang 1991), accompanied by further reduction in memory cost and increase in memory density, will enable an expansion of application domains that lend themselves to table lookup processing. Advances in lookup table size and performance may also lead to the adoption of logarithmic number representation (Chugh and Parhami 2013) in lieu of the currently prevalent floating-point representation, which would lead to additional side benefits.

T

Cross-References

- ▶ [Energy Implications of Big Data](#)
- ▶ [Storage Hierarchies for Big Data](#)
- ▶ [Storage Technologies for Big Data](#)

References

- Chen CLP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inf Sci* 275:314–347

- Chugh M, Parhami B (2013) Logarithmic arithmetic as an alternative to floating-point: a review. In: Proceedings of 47th Asilomar conference on signals, systems, and computers, Pacific Grove, pp 1139–1143
- Cisco Systems (2017) The Zettabyte Era: trends and analysis, White Paper, <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>
- Cray Research (1989) Cray 2 computer system functional description manual, Cray documentation
- Das Sarma D, Matula DW (1995) Faithful Bipartite ROM reciprocal tables. In: Proceedings of 12th symposium on computer arithmetic, Bath, pp 17–28
- De Dinechin F, Tisserand A (2005) Multipartite table methods. *IEEE Trans Comput* 54(3):319–330
- Gabbay F, Mendelson A (1998) Using value prediction to increase the power of speculative execution hardware. *ACM Trans Comput Syst* 16(3):234–270
- Gustafsson O, Johanson K (2006) Multiplierless piecewise linear approximation of elementary functions. In: Proceedings of 40th Asilomar conference on signals, systems, and computers, Pacific Grove, pp 1678–1681
- Hilbert M, Gomez P (2011) The world's technological capacity to store, communicate, and compute information. *Science* 332:60–65
- Jacobson R (2013) 2.5 quintillion bytes of data created every day: how does CPG & retail manage it? IBM Industry Insights, <http://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/>
- Kornerup P, Matula DW (2005) Single precision reciprocals by multipartite table lookup. In: Proceedings of 17th IEEE symposium on computer arithmetic. Cape Cod, pp 240–248
- Lee DU, Luk W, Villasenor J, Cheung PYK (2003) Non-uniform segmentation for hardware function evaluation. In: Proceedings of 13th international conference on field-programmable logic and applications, Lisbon. LNCS, vol 2778. Springer, pp 796–807
- McCallum JC (2017) Graph of memory prices decreasing with time (1957–2017). <http://www.jcmit.net/mem2015.htm>
- Mittal S (2016) A survey of techniques for approximate computing. *ACM Comput Surv* 48(4):62
- Muller J-M (1999) A few results on table-based method. *Reliab Comput* 5:279–288
- Noetzel AS (1989) An interpolating memory unit for function evaluation: analysis and design. *IEEE Trans Comput* 38(3):377–384
- Owens JD et al (2008) GPU computing. *Proc IEEE* 96(5):879–899
- Parhami B (1997) Modular reduction by multi-level table lookup. *Proc 40th Midwest Symp Circuits Syst* 1:381–384
- Parhami B (2005) Computer architecture: from microprocessors to supercomputers. Oxford University Press, New York
- Parhami B (2010) Computer arithmetic: algorithms and hardware designs, 2nd edn. Oxford University Press, New York
- Parhami B, Hung CY (1994) Optimal table lookup schemes for VLSI implementation of input/output conversions and other residue number operations. In: Proceedings of IEEE workshop on VLSI signal processing VII, La Jolla, pp 470–481
- Schulte MJ, Stine JE (1999) Approximating elementary functions with symmetric Bipartite tables. *IEEE Trans Comput* 48(8):842–847
- Smith AJ (1982) Cache memories. *ACM Comput Surv* 14(8):473–530
- Stine JE, Schulte MJ (1999) The symmetric table addition method for accurate function approximation. *J VLSI Signal Process* 21:167–177
- Storer J (1988) Data compression. Computer Science Press, Rockville
- Tang PTP (1991) Table-lookup algorithms for elementary functions and their error analysis. In: Proceedings of symposium on computer arithmetic, Bath, pp 232–236
- Vinnakota B (1995) Implementing multiplication with split read-only memory. *IEEE Trans Comput* 44(11):1352–1356
- White SA (1989) Application of distributed arithmetic to digital signal processing: a tutorial review. *IEEE Trans Acoustics Speech Signal Process* 6(3):4–19
- Zwillinger D (2011) CRC standard mathematical tables and formulae, 32nd edn. CRC Press, Boca Raton

Tachyon

► [Virtual Distributed File System: Alluxio](#)

TARDiS: A Branch-and-Merge Approach to Weak Consistency

Natacha Crooks

The University of Texas at Austin, Austin, TX, USA

Synonyms

[Causal consistency](#); [Distributed systems](#); [Eventual consistency](#)

Definitions

This article targets applications that adopt weaker consistency in large-scale distributed systems in favor of higher availability and better performance.

Overview

In light of the conflicting goals of low latency, high availability, and partition tolerance (Brewer 2000; Gilbert and Lynch 2002), many wide-area services and applications choose to renounce strong consistency in favor of eventual (Vogels 2008) or causal consistency (Ahamad et al. 1994); COPS (Lloyd et al. 2011), Dynamo (DeCandia et al. 2007), Riak (Basho 2017), and Voldemort (2012) are among the many services (Lloyd et al. 2011) that provide a form of weak consistency to applications. Applications that make this decision are often referred to as ALPS applications (**a**vailability, **l**ow **latency**, **p**artition **t**olerance, and **h**igh **s**calability). Distributed ALPS applications, however, are hard to reason about: eventual consistency provides no guarantee on the ordering of operations when these are replicated to other sites. Causal consistency (Ahamad et al. 1994; Lloyd et al. 2011) attempts to mitigate this programming complexity: unlike eventual consistency, causal consistency preserves operation ordering and gives Alice assurance that Bob, whom she has defriended before posting her Spring-break photos, will not be able to access her pictures, even though Alice and Bob access the photo-sharing application using different replicas (Bailis et al. 2012; Cooper et al. 2008; Lloyd et al. 2011).

Nonetheless, programming on top of causal consistency remains complex: causal consistency allows geographically distinct replicas to issue conflicting operations, and the read-write and write-write conflicts that can ultimately result from these operations may cause replicas' states to diverge. To insulate applications from this complexity, systems like COPS (Lloyd et al.

2011) or Dynamo (DeCandia et al. 2007) attempt to preserve the familiar abstraction that an application evolves sequentially through a linear sequence of updates: they aggressively enforce per-object convergence, either through simple deterministic resolution policies or by asking the application to resolve the state of objects with conflicting updates as soon as conflicts arise. These techniques, however, provide no support for meaningfully resolving conflicts between concurrent sequences of updates that involve *multiple* objects: in fact, they often destroy information that could have helped the application in resolving those conflicts. For example, as described further in section “[The Pain Point of Conflicting Write Operations](#)”, deterministic writer-wins (Thomas 1979), a common technique to achieve convergence, hides write-skew from applications, preventing applications from even detecting the anomaly, let alone resolve it. Similarly, exposing multivalued objects without context as in Dynamo (DeCandia et al. 2007) obscures cross-object semantic dependencies.

The rest of this article describes a concrete example of an anomaly that current approaches to merging cannot detect or repair (section “[The Pain Point of Conflicting Write Operations](#)”). It then sketches the design of TARDiS (**t**ransactional **a**synchronously **r**eplicated **d**ivergent **s**tore), a data store explicitly designed for weakly consistent systems that allows for simpler and better merging, through exposing the state of the system as a set of, possibly conflicting, *branches* (section “[A Solution: TARDiS](#)”).

T

Key Research Findings

[The Pain Point of Conflicting Write Operations](#)

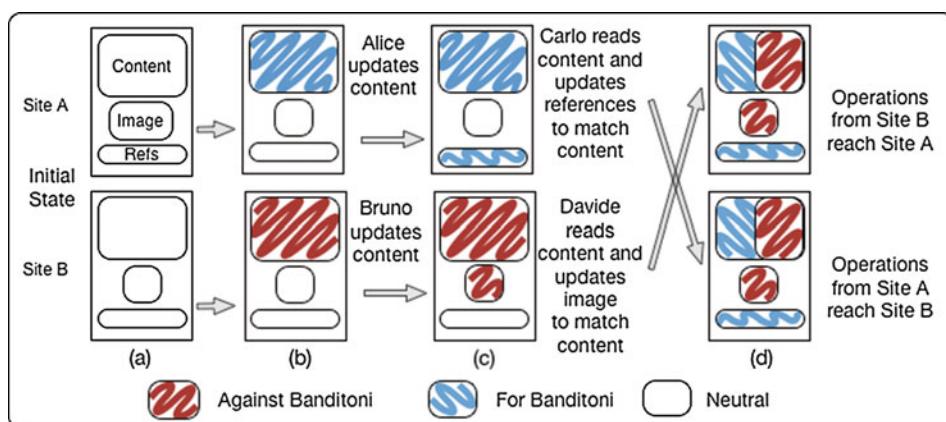
In the classic example used to illustrate the virtues of causal consistency, Alice gains assurance that Bob, whom she had defriended before posting her Spring-break photos, will not be able to access her pictures, even though Alice and Bob access the photo-sharing application using different sites (Bailis et al.

2012; Cooper et al. 2008; Lloyd et al. 2011). ALPS applications, however, face another class of anomalies – write-write conflicts – that causal consistency cannot prevent, detect, or repair.

To illustrate, consider the process of updating a Wikipedia page consisting of multiple HTML objects (Fig. 1a). The page in this example, about a controversial politician, Mr. Banditoni, is frequently modified and is thus replicated on two sites, A and B. Assume, for simplicity, that the page consists of just three objects – the content, references, and an image. Alice and Bruno, who, respectively, strongly support and strongly oppose Mr. Banditoni, concurrently modify the content section of the web page on sites A and B to match their political views (Fig. 1b). Carlo reads the content section on site A, which now favors Mr. Banditoni, and updates the reference section accordingly by adding links to articles that praise the politician. Similarly, Davide reads the update made by Bruno on site B and chooses to strengthen the case made by the content section by updating the image to a derogatory picture of Mr. Banditoni (Fig. 1c). Eventually, the operations reach the other site and, although nothing in the preceding sequence of events violates causal consistency, produce the inconsistent state shown in Fig. 1d: a content section that exhibits a write-write conflict, a reference section in favor of Mr. Banditoni, and an image that is against him.

Worse, there is no straightforward way for the application to detect the full extent of the inconsistency: unlike the explicit conflict in the content sections, the discrepancy between image and references is purely semantic and would not trigger an automatic resolution procedure. This scenario presents an open challenge to current causally consistent systems. These systems struggle to handle such write-write conflicts for two reasons: they choose to syntactically resolve conflicts and do not consider cross-object semantics.

Syntactic conflict resolution. The majority of weakly consistent systems use fixed, syntactic conflict resolution policies to reconcile write-write conflicts (Lloyd et al. 2011; Apache 2017) to maintain the abstraction of sequential storage. Returning to the previous example of the popular merging policy of deterministic writer-wins (DWW): DWW resolves write-write conflicts identically at all sites and ensures that applications never see conflicting writes, which guarantees eventual convergence. In the Wikipedia example, however, guaranteeing eventual convergence would not be sufficient to restore consistency: this policy would choose Bruno's update and ignore the relationship between the content, references, and images of the web page. Such greedy attempt at syntactic conflict resolution is not only inadequate to bridge this semantic gap but in fact can also lose valuable



TARDiS: A Branch-and-Merge Approach to Weak Consistency, Fig. 1 Weakly consistent Wikipedia scenario – a web page replicated on two sites with asyn-

chronous replications. The end state is a write-write conflict on the content and an inconsistent web page

information for reconciliation (here, Alice’s update).

Lack of cross-object semantics. Some systems, like Dynamo (DeCandia et al. 2007) or Bayou (Terry et al 1995), allow for more expressive conflict resolution policies by pushing conflict resolution to the application (DeCandia et al. 2007; Terry et al 1995) but on a per-object basis only. This approach allows for more flexible policies than a purely syntactic solution but reduces conflict resolution to the merging of explicitly conflicting writes. As a result, it is often still overly narrow. For example, it would not detect or resolve inconsistencies that are not write-write conflicts but instead result indirectly from conflicts between two writes, such as the one between the references and the image. Per-object reconciliation policies fail to consider that the effects of a write-write conflict on an object do not end with that object: Carlo and Davide update references and images as they do because they have *read* the conflicting updates to the original content section. Indeed, any operation that depends on one of two conflicting updates is potentially incompatible with all the operations that depend on the other: the shockwaves from even a single write-write conflict may spread to affect the state of the entire database.

There is currently no straightforward way for applications to resolve consistently the kind of multi-object, indirect conflicts that this example illustrates. Transactions (Lloyd et al. 2011, 2013), an obvious candidate, are powerless when the objects that directly or indirectly reflect a write-write conflict are updated, as in the Wikipedia example, by different users. After Bruno’s update, the application has no way to know that Davide’s update is forthcoming: it must therefore commit Bruno’s transaction, forcing Bruno’s and Davide’s updates into separate transactions. Nor would it help to change the granularity of the object that defines the write-write conflict – in the example, by making that object be the entire page. It would be easy to correspondingly scale up the example, using distinct pages that link each other. Short of treating the entire database as the “object,” it is futile to try to define away

these inconsistencies by redrawing the objects’ semantic boundaries.

In essence, conflicting operations fork the entire state of the system, creating distinct *branches*, each tracking the linear evolution of the data store according to a separate thread of execution or site. The Wikipedia, for example, consists of two branches, one in support of Banditoni at site A and one against the politician (site B).

A Solution: TARDiS

As suggested in section “[The Pain Point of Conflicting Write Operations](#)”, conflicting operations fork the entire state of the system, creating distinct *branches*. These branches have traditionally been hidden or greedily merged by systems’ syntactic and per-object resolution strategies, that try, at all cost, to maintain the abstraction of a sequential view of the world. The lack of support for enforcing the cross-object consistency demands expressed in many application invariants thus makes conflict resolution more difficult and error-prone, as the Wikipedia example highlights.

Attempting to isolate applications from the reality of conflicting write operations is therefore, we believe, a well-intentioned fallacy. Conflicting writes “contaminate” data in a way that the storage system cannot understand: application logic is often indispensable to resolve those conflicts. Replicas, however, only see a sequence of read/write operations and are unaware of the application logic and invariants that relate these operations. The storage system should avoid deterministic quick fixes and instead give applications the information they need to decide what is best. The question becomes: how can one provide applications with the best possible system support when merging conflicting states?

TARDiS (Crooks et al. 2016) attempts to provide an answer to this question. TARDiS is an asynchronously replicated, multi-master key-value store designed for applications built above weakly consistent systems. TARDiS renounces the one-size-fits-all abstraction of sequential storage and instead exposes applications, when appropriate, to concurrency and distribution. TARDiS’ design is predicated on a simple notion:

to help developers resolve the anomalies that arise in such applications, each replica should faithfully store the full context necessary to understand how the anomalies arose in the first place but only expose that context to applications when needed. By default in TARDiS, applications execute on a branch and hence perceive storage as sequential. But when anomalies arise, TARDiS reveals to the users the intricate details of distribution. TARDiS hence gives applications the flexibility of deciding if, when, and how divergent branches should be merged.

TARDiS provides two novel features that simplify reconciliation. First, it exposes applications to the resulting independent branches and to the states at which the branches are created (fork points) and merged (merge points). Second, it supports atomic merging of conflicting branches and lets applications choose when and how to reconcile them. Branches, together with their fork and merge points, naturally encapsulate the information necessary for semantically meaningful merging: they make it easy to identify all the objects to be considered during merging and pinpoint when and how the conflict developed. This context can reduce the complexity and improve the efficiency of automated merging procedures, as well as help system administrators when user involvement is required. Returning to the Wikipedia example, a Wikipedia moderator presented with the two conflicting branches would be able to reconstruct the events that led to them and handle the conflicting sources according to Wikipedia’s guidelines (Wikipedia 2017). Note that merging need not simply involve deleting one branch. Indeed, branching and merging states enables merging strategies with richer semantics than aborts or rollbacks (Sovran et al. 2011). It is TARDiS’s richer interface that gives applications access to content that is essential to reasoning about concurrent updates, reducing the complexity of programming weakly consistent applications. In many ways, TARDiS is similar to Git (2017): users operate on their own branch and explicitly request (when convenient) to see concurrent modifications, using the history recorded by the underlying branching storage to help them resolve conflicts.

Unlike Git, however, branching in TARDiS does not rely on specific user commands but occurs implicitly, to preserve availability in the presence of conflicts. Two core principles underpin TARDiS: branch-on-conflict and inter-branch isolation. Branch-on-conflict lets TARDiS logically fork its state whenever it detects conflicting operations and store the conflicting branches explicitly. Inter-branch isolation guarantees that the storage will appear as sequential to any thread of execution that extends a branch, keeping application logic simple. The challenge is then to develop a data store that can keep track of independent execution branches, record fork and merge points, facilitate reasoning about branches, and, as appropriate, atomically merge them – while keeping performance and resource overheads comparable to those of weakly consistent systems.

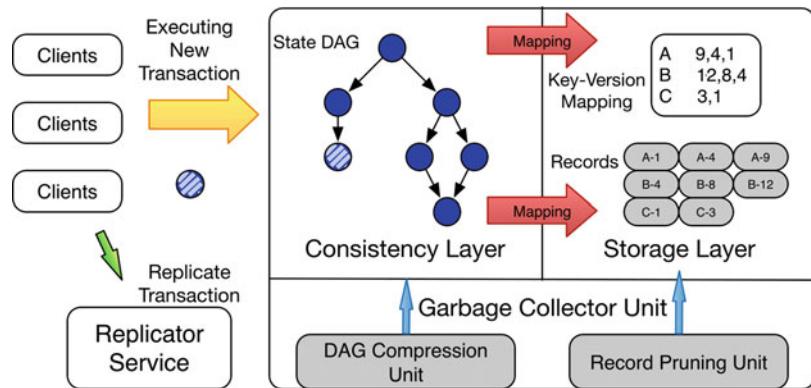
TARDiS uses multi-master asynchronous replication: transactions first execute locally at a specific site and are then asynchronously propagated to all other replicas Fig. 2. Each TARDiS site is divided into two components – a *consistency layer* and a *storage layer*:

- *Consistency layer.* The consistency layer records all branches generated during an execution with the help of a directed acyclic graph, the *state DAG*. Each vertex in the graph corresponds to a logical state of the data store; each transaction that updates a record generates a new state.
- *Storage Layer.* The storage layer stores records in a disk-based B-tree. TARDiS is a multiversioned system: every update operation creates a new record version.

Much of TARDiS logic is geared toward efficiently mapping the consistency layer to the storage layer and maintaining low metadata overhead. Two techniques are central to the system’s design – *DAG compression* and *conflict tracking*:

- *DAG compression.* TARDiS tracks the minimal information needed to support branch

TARDiS: A Branch-and-Merge Approach to Weak Consistency, Fig. 2
TARDiS architecture



merges under finite storage. It relies on a core observation: most merging policies only require the fork points and the leaf states of a given execution. All intermediate states can be safely removed from the state DAG, along with the corresponding records in the storage layer.

- *Conflict tracking.* To efficiently construct and maintain branches, TARDiS introduces the notion of conflict tracking. TARDiS summarizes branches as a set of fork points and merge points only (a *fork path*) and relies on a new technique, *fork point checking* to determine whether two states belong to the same branch. Fork paths capture *conflicts* rather than dependencies. As such, they remain of small size (conflicts represent a small percentage of the total number of operations) and allow TARDiS to track concurrent branches efficiently while limiting memory overhead. This is in contrast to the traditional dependency checking approach (Du et al. 2014; Lloyd et al. 2011; Mahajan et al. 2011), which quickly becomes a bottleneck in causally consistent systems (Bailis et al. 2012; Du et al. 2014).

This article does not attempt to further discuss the details of these techniques and defer the reader to the corresponding SIGMOD paper (Crooks et al. 2016). Generally though, the system is promising: using TARDiS, for instance, rather than BerkeleyDB (Olson et al. 1999) to implement CRDTs (Shapiro et al. 2011) – a

library of scalable, weakly consistent datatypes – cuts code size by half and improves performance by four to eight times.

Future Directions for Research

Branches are a useful abstraction for reasoning about weakly consistent systems and implementing such systems. Can they be useful in other settings? Strongly consistent geo-distributed systems face many of the same challenges of weakly consistent systems. Concurrent conflicting writes at different replicas, for instance, still arise in strongly consistent systems, though they are hidden from the application. Can the cost of synchronizing across replicas on every write operation be mitigated by using branches? Moreover, many consistency definitions like causal consistency, snapshot isolation (Berenson et al. 1995), or parallel snapshot isolation (Sovran et al. 2011) allow, implicitly, for a small degree of branching (the ability to read from a stale state can be viewed as a limited form of branching). Can we simplify the formulation of these guarantees by using branches?

References

- Ahamad M, Neiger G, Burns J, Kohli P, Hutto P (1994) Causal memory: definitions, implementation and programming. Technical report, Georgia Institute of Technology
Apache (2017) Cassandra. <http://cassandra.apache.org/>

- Bailis P, Fekete A, Ghodsi A, Hellerstein JM, Stoica I (2012) The potential dangers of causal consistency and an explicit solution. In: Proceedings of the 3rd ACM symposium on cloud computing, SOCC '12, pp 22:1–22:7. <http://doi.acm.org/10.1145/2391229.2391251>
- Basho (2017) Riak. <http://basho.com/products/>
- Berenson H, Bernstein P, Gray J, Melton J, O'Neil E, O'Neil P (1995) A critique of ANSI SQL isolation levels. In: ACM SIGMOD record, vol 24, pp 1–10
- Brewer EA (2000) Towards robust distributed systems (abstract). In: Proceedings of the 19th ACM symposium on principles of distributed computing, PODC '00. <http://doi.acm.org/10.1145/343477.343502>
- Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. Proc VLDB Endow 1(2):1277–1288
- Crooks N, Pu Y, Estrada N, Gupta T, Alvisi L, Clement A (2016) Tardis: a branch-and-merge approach to weak consistency. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. ACM, New York, pp 1615–1628. <http://doi.acm.org/10.1145/2882903.2882951>
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. In: Proceedings of 21st ACM symposium on operating systems principles, SOSP '07, pp 205–220. <http://doi.acm.org/10.1145/1294261.1294281>
- Du J, Iorgulescu C, Roy A, Zwaenepoel W (2014) Gentleain: cheap and scalable causal consistency with physical clocks. In: Proceedings of the ACM symposium on cloud computing, SOCC '14, pp 4:1–4:13. <http://doi.acm.org/10.1145/2670979.2670983>
- Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59. <http://doi.acm.org/10.1145/564585.564601>
- Git (2017) Git: the fast version control system. <http://git-scm.com>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In: Proceedings of the 23rd ACM symposium on operating systems principles, SOSP '11, pp 401–416. <http://doi.acm.org/10.1145/2043556.2043593>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Proceedings of the 10th USENIX symposium on networked systems design and implementation, NSDI '13, pp 313–328. <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/lloyd>
- Mahajan P, Setty S, Lee S, Clement A, Alvisi L, Dahlin M, Walfish M (2011) Depot: cloud storage with minimal trust. ACM Trans Comput Syst 29(4):12
- Olson MA, Bostic K, Seltzer M (1999) Berkeley DB. In: Proceedings of the annual conference on USENIX annual technical conference, ATEC '99. <http://dl.acm.org/citation.cfm?id=1268708.1268751>
- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) A comprehensive study of convergent and commutative replicated data types. Rapport de recherche RR-7506, INRIA
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceedings of the 23rd ACM symposium on operating systems principles, SOSP '11, pp 385–400. <http://doi.acm.org/10.1145/2043556.2043592>
- Terry DB, Theimer MM, Petersen K, Demers AJ, Spreitzer MJ, Hauser CH (1995) Managing update conflicts in Bayou, a weakly connected replicated storage system. In: Proceedings of the 15th ACM symposium on operating systems principles, SOSP '95, pp 172–182. <http://doi.acm.org/10.1145/224056.224070>
- Thomas RH (1979) A majority consensus approach to concurrency control for multiple copy databases. ACM Trans Database Syst 4(2):180–209. <http://doi.acm.org/10.1145/320071.320076>
- Vogels W (2008) Eventually consistent. Queue 6(6): 14–19. <http://doi.acm.org/10.1145/1466443.1466448>
- Voldemort (2012) Project Voldemort: a distributed database. Online, <http://project-voldemort.com/>
- Wikipedia (2017) Wikipedia: conflicting sources. http://en.wikipedia.org/wiki/Wikipedia:Conflicting_sources

Target Environment

▶ System Under Test

Test

▶ Microbenchmark

Test Harness

▶ Benchmark Harness

Test Object

▶ System Under Test

Tested Environment

► System Under Test

Time Series Prediction

► Big Data Analysis Techniques

Tools and Libraries for Big Data Analysis

Shadi Khalifa

Queen's University, Kingston, Canada

Overview

With almost everything now online, organizations look at the Big Data collected to gain insights for improving their services. The Big Data analysis pipeline consists of a number of processes requiring different tools. The Big Data analysis pipeline processes can be summarized as follows:

- *Data Exploration*: analysts go through the data, using ad hoc queries and visualizations, to better understand the data.
- *Data Engineering*: analysts clean, prepare, and transform the data for modeling using batch processing to run computational and input/output (IO) intensive operations.
- *Modeling/Scoring*: data models are trained, using iterative processing, on the prepared data, and trained models are used to score the unlabeled data.

For each Big Data analysis process, using the right tool(s) can significantly reduce the processing time and would generate better insights. Some of the still-useful tools are those inherited from the pre-Big Data era, where data could fit in a single node memory. These tools provide a huge set of algorithms, but they do not scale to meet the Big Data requirements.

To handle Big Data, tool developers went into two separate ways. Some developed new scalable tools designed to handle Big Data. Others went after making pre-Big Data era tools parallel, distributed, and scalable to handle Big Data and benefit from what these tools have to offer.

In this chapter, the most popular Big Data analysis tools and libraries are presented, and their limitations are highlighted. However, not all tools are covered in this chapter due to space limitations. Interested readers can check our survey paper (Khalifa et al. 2016) for a more comprehensive list of tools and libraries.

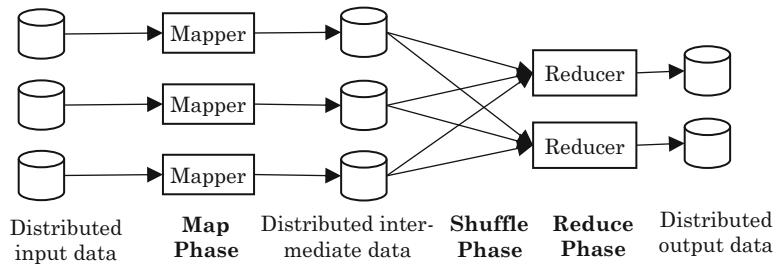
The Big Data Era Tools

To solve the scalability problem, *Google* introduced the *MapReduce* (Dean and Ghemawat 2004) platform to manage and process large heterogeneous datasets. Following that, *Yahoo!* developed *Hadoop* (White 2009), an open-source implementation of MapReduce and later incubated by *Apache*, allowing everyone to benefit from it. *Apache* also incubated the *Spark* (Zaharia et al. 2010) project for in-memory computations, which is now the main rival to *Hadoop*.

MapReduce, as presented in Fig. 1, consists of *Map*, *Shuffle*, and *Reduce* phases, which are executed sequentially, utilizing all nodes in the cluster. In the *Map* phase, the programmer-provided *Map* function (*Mapper*) processes the input data and outputs intermediate data in the form of $\langle key, value \rangle$ tuples which get stored on disk. The *Shuffle* phase then groups values to the same key together and sends them to the reduce nodes over the network. Finally, the programmer-provided *Reduce* function (*Reducer*) reads the intermediate data from disk, processes it, and generates the final output. The *Map/Reduce* functions are executed in parallel over a set of distributed data files in a single program multiple data (SPMD) paradigm.

Apache Spark follows the *MapReduce* paradigm except that it runs with one *reducer*, which can be a bottleneck. *Spark* uses read-only Resilient Distributed Datasets (RDDs) to

Tools and Libraries for Big Data Analysis, Fig. 1
MapReduce dataflow



```
CREATE TABLE <model_name> AS
SELECT <label>, cast(<<feature>> AS <<datatype>>) AS feature
FROM {
    SELECT TRAIN_MODEL(<<feature>>,<label>) AS <label>,<<feature>>,
           <<weight>>,<<covar>>
    FROM <training_dataset_table> ) t
GROUP BY <label>,<<feature>>;
```

Tools and Libraries for Big Data Analysis, Fig. 2 Training using Hivemall

```
CREATE VIEW <prediction_table_name>
AS SELECT <rowid>, <score>, <label> FROM (
    SELECT <t.rowid>, <m.label>, <m.score>
    FROM <testing_dataset_table> t LEFT OUTER JOIN
        <model name> m ON (<<t.feature>> = <<m.feature>>)) ;
```

Tools and Libraries for Big Data Analysis, Fig. 3 Prediction using Hivemall

provide in-memory support for iterative jobs. RDDs can also reconstruct themselves in case of failure to provide fault tolerance without disk checkpointing. *Spark*, however, is restricted to the size of available memory and has to reconstruct the RDDs from disk if it runs out of memory. *Spark* does not run *Hadoop* jobs; however, they can coexist on the same cluster using *Apache Mesos* (Hindman et al. 2011).

A number of libraries have been developed to provide a scalable distributed implementation of the analytics algorithms, so that analysts do not have to write the MapReduce procedures themselves. *Apache Mahout* is a scalable distributed machine learning library, where the machine learning algorithms are implemented as a sequence of MapReduce jobs with an outside driver program to control loop execution. *Apache Pig* uses the *Pig Latin* language (Olston et al. 2008) to abstract coding MapReduce analytics jobs without using the Java MapReduce idioms. *Apache Hive* (Thusoo et al. 2009) uses *HiveQL*, an SQL-like query language, to create MapReduce batch jobs. *Hivemall* (Yui and Kojima 2013)

extends *HiveQL* with a scalable machine learning library. Thus, machine learning models can be trained (Fig. 2) and scored (Fig. 3) from within the *HiveQL* SQL-like statements. The *HiveQL* queries are then translated to MapReduce batch jobs for execution.

For *Spark*, *Spark SQL* (previously known as *Shark*) (Xin et al. 2013) allows using SQL statements to query structured data inside Spark programs. *MLBase* (Talwalkara et al. 2012) uses the *Mlib* (Sparks et al. 2013) library to provide distributed machine learning at scale on *Spark*. *Cloudera Oryx* forks from *Apache Mahout* to run on *Spark*. *Oxdata H2O* provides in-memory machine learning and predictive analytics using *Spark*. *Deeplearning4j* provides deep learning algorithms implementation on *Hadoop* and *Spark*.

The introduced libraries provide an easy-to-use scalable distributed implementation of the analytics algorithms; they require none to minimum MapReduce knowledge and are optimized for parallel processing.

MapReduce is a popular batch processing approach because of its scalability, fault tolerance,

ease to program, and flexibility. However, batch processing is designed to execute one or more jobs without manual intervention. This makes it perfect for scenarios where a program needs to have a single run on a huge dataset. However, not all Big Data analysis tasks can be efficiently executed on MapReduce.

Stream Analytics Tools

Unlike MapReduce that is designed to analyze data at rest, stream analysis tools analyze data in motion where the utility of data points declines quickly. Some use cases are analyzing monitoring logs, sensor network feeds, and high-volume news feeds like twitter. Stream Analytics can be in the form of *stream processing* or *micro-batch processing*.

Stream processing provides low latency as data is analyzed as soon as it arrives. However, it is technically challenging to support certain types of analytics like time series analysis since the analysis runs on the records as they arrive and not wait for the complete transaction.

Apache Storm is a distributed streaming solution that can be used with any programming language and can scale to massive numbers of messages per node per second. After the analyst designs the analysis pipeline, *Storm* uses a pull model where the receiving nodes pull the data when they can process it.

Apache Samza is another open-source distributed stream processing framework. It comes as part of *Hadoop2.0 (YARN)* and uses *Apache Kafka* publish/subscribe messaging system to guarantee that data is processed in the order it arrives and that no data is ever lost.

Google has the *MillWheel* (Akidau et al. 2013) in-house system that allows analysts to create stream graphs and define the application code for each graph node, while it handles the continuous data flow, data persistence, and failure recovery.

IBM InfoSphere Streams is a component of the *IBM* analytics solution. It supports analyzing data continuously at high rates using real-time machine learning. *InfoSphere Streams* supports runtime modifications, where input streams, output streams, and operations can be added or removed dynamically without restarting the system. It also

allows embedding user-defined Java and C++ analytics routines.

Micro-batch processing presents a middle ground between streams and batch processing. It has higher latency than streams as it buffers the input and only processes it when the buffer is full. However, micro-batching is less technically challenging as it allows the use of existing batch algorithms.

Using this approach is *Yahoo! Nova* (Olston et al. 2011) which allows analysts to create workflows of *Pig* programs to process continually arriving data. Also, *Spark Streaming* extends *Spark* to allow joining stream data with historical data using the same *Spark* code written for batch processing.

Visual Analytics Tools

With Big Data, analysts can drown in the excessive volumes of data. This can lead to analyzing the wrong or incomplete set of attributes or becoming frustrated with the whole analytics process. Visualization solutions are designed for business analysts to allow them to have an interactive conversation with their data.

IBM Watson Analytics (Rais-Ghasem et al. 2013) allows analysts to go from data to analysis in seconds without any setup or configuration. It allows analysts to use visualization and natural language to understand their data. It relies on *IBM SPSS Analytics Server* and *IBM Big Insights* to automatically build and use data models. On the down side, *Watson Analytics* requires data to be pre-cleaned outside the *Watson* framework. It does not output an analytic model to use for further analysis, and it does not specify how the analysis outputs were achieved making it hard to trust the outputs.

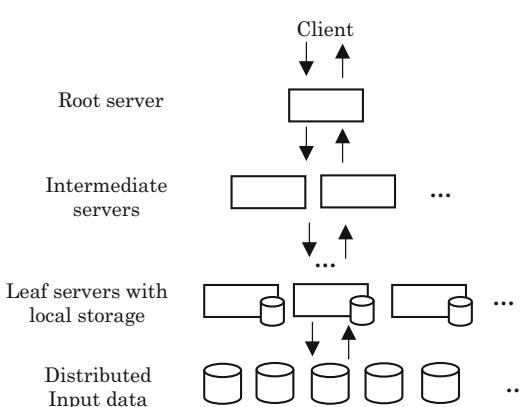
SAS Visual Analytics powered by the *SAS Analytics framework* is designed to empower business analysts with limited technical skills to do analytics using *Hadoop* without any programming. In addition to data exploration, analysts can do sophisticated analytics like forecasting using a drag-and-drop approach.

MicroStrategy and *Tableau* allow analysts to access data from multiple sources (spreadsheets, databases, or distributed filesystems) and present them in interactive visualizations for analysis. The analytics capabilities of those solutions are limited as they do not implement machine learning techniques. Their power comes from their availability on mobile devices and allowing analysts to share visualizations.

Exploratory Analytics Tools

The frequent writing to disk and the extensive communication between nodes in the MapReduce shuffle phase to support fault tolerance hinder efficient support for interactive tasks required in the data exploration phase. Some solutions are proposed to support interactive exploratory analytics.

Google Dremel (Melnik et al. 2010) (aka *Google BigQuery*) leverages massive parallel processing, nested data modeling, and columnar storage to improve retrieval efficiency for interactive analytics scenarios. *Dremel* is a *Google* proprietary solution that executes its query in parallel without being translated to a sequence of MapReduce jobs. *Dremel* uses multilevel tree processing (Fig. 4). Leaf servers only scan the needed columns in parallel. Intermediate servers carry out parallel aggregation on the scanned data. Finally, the root server aggregates the intermediate results.



Tools and Libraries for Big Data Analysis, Fig. 4
Dremel query execution flow

However, with columnar storage the number of columns accessed affects performance. *Google* has introduced *PowerDrill* (Hall et al. 2012) to have data in-memory for faster computations. However, this makes it constrained to the available memory.

Dremio, *Apache Drill*, and *Cloudera Impala* offer open-source implementations of *Google Dremel*. They support querying and joining data of different formats from different sources. These solutions support dynamic schema discovery, where users can define the schema (column name, data type, length) in the query or let the tools discover the schema from the metadata for self-describing data formats like JSON, thus, no need for the Extract-Transform-Load operations for consolidating data from multiple silos. They also provide a JDBC/ODBC interface to connect to Business Intelligence and visualization tools. On the downside, they adopt an optimistic execution model and do not persist intermediate data, making them fault-intolerant.

In the data exploration phases, there are a lot of trial and error of different operations. Processing the entire dataset every time can be very time consuming with Big Data. To speed up this process, a set of tools were developed to provide a quick retrieval of approximate results to help analysts decide if an operation is useful or not.

The *Early Accurate Result Library (EARL)* (Laptev et al. 2012) extends *Hadoop* to allow early termination and incremental computation, along with providing an online indicator to estimate the achieved accuracy so far. *EARL* provides early approximate results by iterating over data samples and aggregating the results until an acceptable accuracy is reached.

BlinkDB (Agrawal et al. 2013) extends *Hive* to provide fast approximate results with statistical error guarantees. *BlinkDB* uses a dynamic sample selection strategy to select an appropriately sized sample based on the desired query accuracy or response time. *BlinkDB* maintains a set of precomputed and carefully chosen data samples, so that when a query arrives, it can be directly executed on the appropriate sample. Samples in *BlinkDB* are chosen using an optimization formula that considers the data distribution, past queries, stor-

age constraints, and several other system-related factors.

Pre-Big Data Era Tools

The pre-Big Data era tools are thesis still-useful tools inherited from the pre-Big Data era, where data could fit in a single node memory. These tools provide a huge set of algorithms, but they do not scale to meet the Big Data requirements. With the growing popularity of *MapReduce*, the pre-Big Data era tools started adding support for *MapReduce* to enhance their scalability.

For a lot of people, doing analysis means using *Microsoft Excel*. Thus, *Microsoft* extended *Excel* with *Excel Analytics (Power Query, Data Analysis Expressions (DAX) Language, Data Mining add-in)*, where analysts can use the *Power Query* Sheet GUI to fetch and merge data from different sources and do data preparation and transformation. The *DAX* language keeps track of all executed steps to support undos, and the *Data Mining add-in* sends the prepared data to the SQL server for modeling. *Power Query* is limited to a maximum of 1,000,000 records per dataset. For that, *Microsoft* offers *Microsoft Tabular*, a server-based solution for *in-database* analytics on structured data. *Microsoft* also offers *Daytona* (Barga et al. 2012) to offload *Excel*'s operations to *MapReduce* on the cloud for distributed processing.

Google OpenRefine is a browser-based, spreadsheet-style tool designed for data exploration and preparation but does not support data modeling. Unlike *Excel's Power Query*, *OpenRefine* supports interactivity and can handle any number of records by only showing a data sample to analysts. However, *OpenRefine* still runs on a single machine and is thus limited to the machine's memory and computational resources.

IBM Cognos provides a simple GUI for manipulating data in the form of spreadsheets and data cubes. As with previous solutions, *Cognos* runs on a single machine, making it limited to small and medium datasets. For that, *IBM* offers the *BigInsights BigSheets* which is a browser-based, spreadsheet-style tool in the *IBM InfoS-*

phere BigInsights solution that enables business analysts to explore, manipulate, and analyze Big Data using the underlying *Hadoop* layer for distributed processing.

IBM also offers the *SPSS Modeler* which supports a wide range of advanced algorithms and techniques for text analytics, decision management, predication, and optimization. The *SPSS Modeler* provides a drag-and-drop interface for the analytics operations where almost no programming is required. While the *SPSS Modeler* runs on a single machine, the *IBM Analytic Server* extends it to run scalable distributed analytics on *Hadoop*.

A yet another very popular drag-and-drop solution is *RapidMiner*. It supports sub-workflows, wizards, and quick fixes extensively, making complex workflows easier to design and interpret. *Radoop* (Prekopsak et al. 2011) extends *RapidMiner* to run on *Hive*, *MLib*, and *Mahout* to support Big Data analysis. Also, the *streams library* plug-in (Bockermann and Blom 2012) adds stream analytics capabilities to *RapidMiner*. It provides generic streaming wrappers of the *RapidMiner* operations to make them run on partially available data and feed them the rest as it arrives.

For programming savvies, there are a number of popular analytics programming languages and application programming interfaces (APIs). There is *R* made by and for statisticians and *Weka* made for data miners. While these tools provide a large set of algorithms, they only run on a single machine, making them unsuitable to use with Big Data. *RHadoop* extends *R* and allows running *R* code on *Hadoop*. *DistributedWekaHadoop* and *DistributedWekaSpark* (Koliopoulos et al. 2015) packages extend *Weka* to access *Hadoop* and *Spark*, respectively.

Recently, *Python* has become one of the most widely used languages for analytics. A large number of analytics libraries have been developed for *Python*, like *scikit-learn* and *TensorFlow*. This creates a single environment, where analysts can do both, general-purpose programming and analytics. Following the same trend, *Microsoft* offers *F#*, a cross-platform functional language. *F#* provides libraries for fetching data

from different sources and allows analysts to use the .NET machine learning libraries to do analytics with simple code. Programmers, however, need to write their own MapReduce functions to execute their code in parallel.

Future Big Data Analysis Tools

There has been a continuous improvement in analysis tools to address different analytics scenarios; however, there are still some gaps. In this section, we highlight some principles for future Big Data analysis tools to consider. Following these principles would improve the tools and allow them to broader support different analytics scenarios.

1. *Extensibility.* Tools can become obsolete if they are designed to only support a fixed number of data stores and analytics techniques. It is thus crucial for future tools to have a plug-in architecture to support adding new algorithms and data stores with minimal modifications to the core code.
2. *Distributed processing.* To handle Big Data, future tools need to seamlessly distribute the processing among many nodes with minimum to no user involvement.
3. *Approximate processing.* While designing the analytics process, users are not usually sure of which operation to use and how it is going to affect the results. This makes them try different operations, which can be time consuming if they run on the whole dataset. Thus, future tools need to provide quick retrieval of approximate results from a small representative sample to give users an indication of an operation's results without running it on the entire dataset.
4. *Execution and storage optimization.* Having the data distributed among cloud nodes requires optimizing the analytics execution to utilize the shared resources while minimizing data movement. Future tools need to predict what data will be needed by future operations and make sure that this data is available on the underutilized nodes for the future operations.

They also need to provide an abstraction layer to hide the heterogeneity of the data stores so that users can seamlessly join data from multiple sources with minimum data movement.

5. *Fault tolerance.* Big Data analysis can run for long periods. Having to restart from the beginning in case of failure is not acceptable. Future tools need to support fault tolerance while minimizing its impact on performance. Fault tolerance can be in the form of *validation* that includes checking input compatibility to operations and that computational and storage resources are available prior to execution. It can also be in the form of *masking operation failure* by trying to re-execute it, executing it on an alternate resource, using checkpoints to save processed data, executing an alternate implementation, executing multiple redundant copies, and executing user-defined exception handling methods.
6. *Intelligent assistance.* With the huge set of available analytics techniques, even experts sometimes need help. Intelligent assistance can aid users in choosing and configuring the analytics operations based on the input dataset and the analysis goals. Assistance can range from recommending operations to generating full analytics workflows. Providing intelligent assistance in future tools is important for analytics to be more accessible, to minimize the time-to-insights, and to enhance the quality of analytics.
7. *Multiple user interfaces.* For future tools, a combination of user interfaces should be provided to meet the needs of users of different skillsets. They should provide the flexibility and the extensibility of the programming languages, the ease of use of drag-and-drop interfaces, the ad hoc capabilities of the SQLs, and the familiar environment of the spreadsheets.

References

- Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European conference on computer systems, Prague, 14–17 Apr 2013, pp 29–42

- Akida T, Balikov A, Bekiroğlu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) MillWheel: fault-tolerant stream processing at internet scale. *Proc VLDB Endow* 6(11):1033–1044
- Barga R, Ekanayake J, Wei L (2012) Project Daytona: data analytics as a cloud service. In: Proceedings of the IEEE 28th international conference on data engineering, Washington, 1–5 Apr 2012, pp 1317–1320
- Bockermann C, Blom H (2012) Processing data streams with the rapidminer streams-plugin. In: Proceedings of the rapidminer community meeting and conference, Budapest, 28–31 Aug 2012
- Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on operating systems design & implementation, San Francisco, 6–8 Dec 2004
- Hall A, Bachmann O, Büssow R, Gănceanu S, Nunkesser M (2012) Processing a trillion cells per mouse click. *Proc VLDB Endow* 5(11):1436–1446
- Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX conference on networked systems design and implementation, Boston, 30 Mar–1 Apr 2011, pp 295–308
- Khalifa S, Elshater Y, Sundaravarathan K, Bhat A, Martin P, Imam F, Rope D, Mcroberts M, Statchuk C (2016) The six pillars for building big data analytics ecosystems. *ACM Comput Surv* 49:2. Article 33
- Koliopoulos A-K, Yiapanis P, Tekiner F, Nenadic G, Keane J (2015) A parallel distributed Weka framework for big data mining using spark. In: Proceedings of IEEE international congress on big data, New York City, 27 June–2 July 2015, pp 9–16
- Laptev N, Zeng K, Zaniolo C (2012) Early accurate results for advanced analytics on MapReduce. *Proc VLDB Endow* 5(10):1028–1039
- Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T (2010) Dremel: interactive analysis of web-scale datasets. *Proc VLDB Endow* 3(1–2):330–339
- Olston C, Reed B, Srivastava U, Kumar R, Tomkins A (2008) Pig latin: a not-so-foreign language for data processing. In: Proceedings of the ACM SIGMOD international conference on management of data, Vancouver, 10–12 June 2008, pp 1099–1110
- Olston C, Chiou G, Chitnis L, Liu F, Han Y, Larson M, Neumann A, Rao VBN, Sankarasubramanian V, Seth S, Tian C, ZiCornell T, Wang X (2011) Nova: continuous Pig/Hadoop workflows. In: Proceedings of the ACM SIGMOD international conference on management of data, Athens, 12–16 June 2011, pp 1081–1090
- Prekopcsak Z, Makrai G, Henk T, Gaspar-Papanek C (2011) Radoop: analyzing big data with rapidminer and hadoop. In: Proceedings of RCOMM 2011, Dublin, 7–10 June 2011
- Rais-Ghasem M, Grossot R, Petitclerc M, Wei Q (2013) Towards semantic data analysis. In: Proceedings of the 2013 CASCON, Markham, 18–20 Nov 2013, pp 192–199
- Sparks E, Talwalkar A, Smith V, Pan X, Gonzales J, Kraska T, Jordan M, Franklin MJ (2013) MLlib: an API for distributed machine learning. In: Proceedings of the 2013 IEEE 13th international conference on data mining, Dallas, 7–10 Dec 2013, pp 1187–1192
- Talwalkara A, Kraska T, Griffith R, Duchi J, Gonzalez J, Britz D, Pan X, Smith V, Sparks E, Wibisono A, Franklin MJ, Jordan MI (2012) MLbase: a distributed machine learning wrapper. In: Proceedings of big learning workshop at NIPS, Lake Tahoe, 7–8 Dec 2012
- Thusoo A, Sarma JS, Jain N, Zheng S, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R (2009) Hive: a warehousing solution over a map-reduce framework. *Proc VLDB Endow* 2(2):1626–1629
- White T (2009) Hadoop: the definitive guide (1st edn). O'Reilly Media, Sebastopol
- Xin RS, Rosen J, Zaharia M, Franklin MJ, Shenker S, Stoica I (2013) Shark: SQL and rich analytics at scale. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, New York City, 23–28 June 2013, pp 13–24
- Yui M, Kojima I (2013) A database-hadoop hybrid approach to scalable machine learning. In: Proceedings of the 2013 IEEE international congress on big data, Santa Clara, 27 June–2 July 2013, pp 1–8
- Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing, Boston, 22–25 June 2010

TPC

Meikel Poess¹ and Raghu Nambiar²

¹Server Technologies, Oracle Corporation, Redwood Shores, CA, USA

²Advanced Micro Devices (AMD), Suwanee, GA, USA

T

Synonyms

Performance Consortia; Transaction Processing Performance Council

Definitions

A nonprofit corporation founded to define transaction processing and database benchmarks and

to disseminate objective, verifiable TPC performance data to the industry.

Historical Background

Originally formed in 1988, the Transaction Processing Performance Council (TPC) is a non-profit corporation focused on defining database processing benchmarks and disseminating objective, verifiable performance data to the IT industry. The TPC was originally founded in response to a growing trend at the time, known as “benchmarketing.” Effectively, this was a common practice of vendors publishing unverifiable claims based on their own price and performance data in order to present their systems in the best possible way to improve the market’s perception of their product.

“Benchmarketing” effectively enabled vendors to exaggerate performance and even reliability claims in order to boost sales. The need for a vendor-neutral standards organization that focused on creating and administering fair and comprehensive benchmark specifications to objectively evaluate database systems under demanding but consistent and comparable workloads quickly became apparent. Several influential database academics and industry leaders began working to establish an organization charged with leading the effort to impose order and consistency to the process of benchmarking products fairly and objectively – this effort ultimately culminated in the formation of the TPC.

Foundations

The TPC has a long history in benchmarks. Figure 1 shows the current TPC benchmark timeline. Its benchmarks are used by hardware and software providers for internal and external purposes. Internally, they are used during product development to measure the performance of existing and planned products. Externally, they are used to provide reliable, verifiable, comparable, and relevant performance data to

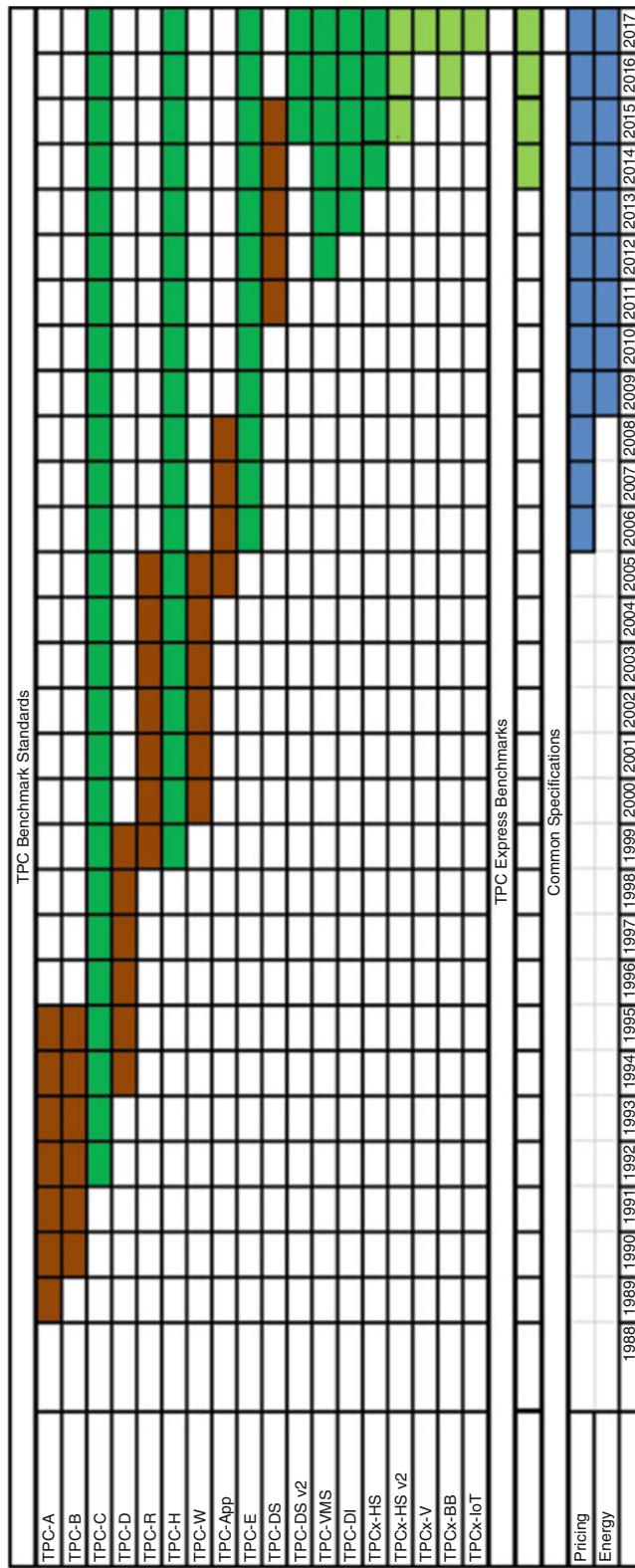
advertise their products. Customers use TPC performance data to make purchase decisions.

To date the TPC has approved a total of 18 benchmark standards and 2 common specifications to measure energy and guarantee fair pricing for benchmarked systems. Of the 18 benchmark standards, TPC-E [19, 21], TPC-C [17], TPC-H (Nambiar and Poess 2006), TPC-DS (Poess et al. 2002; Poess et al. 2007; Nambiar and Poess 2006; Poess et al. 2017), TPC-DI (Poess. et al. 2014), TPC-VMS, TPCx-V, TPCx-BB, TPCx-HCI, and TPCx-IoT (Poess et al. 2018) are currently active standards.

Benchmark History

TPC-C and TPC-E are online transaction processing (OLTP) benchmarks. **TPC-C** simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment but, rather, represents any industry that must manage, sell, or distribute a product or service. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed online or queued for deferred execution. It does so by exercising a breadth of system components associated with such environments, which are characterized by:

- The simultaneous execution of multiple transaction types that span a breadth of complexity
- Online and deferred transaction execution modes
- Multiple online terminal sessions
- Moderate system and application execution time
- Significant disk input/output



TPC, Fig. 1 Current TPC benchmarks

- Transaction integrity (ACID properties)
- Nonuniform distribution of data access through primary and secondary keys
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
- Contention on data access and update

TPC-C performance is measured in new-order transactions per minute. The primary metrics are the transaction rate (tpmC), the associated price per transaction (\$/tpmC), and the availability date of the priced configuration.

TPC-E is an enterprise-class online transaction processing (OLTP) benchmark. It is more complex than previous OLTP benchmarks such as TPC-C because of its diverse transaction types, more complex database, and overall execution structure. TPC-E involves a mix of 12 concurrent transactions of different types and complexity, either executed online or triggered by price or time criteria. The database is comprised of 33 tables with a wide range of columns, cardinality, and scaling properties. TPC-E is measured in transactions per second (tpsE). While the benchmark portrays the activity of a stock brokerage firm, TPC-E is not limited to the activity of any particular business segment but rather represents any industry that must report upon and execute transactions of a financial nature.

TPC-H (Pöss and Floyd 2000) is an enterprise-class decision support benchmark. It consists of a suite of business-oriented ad hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size) and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing

power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The TPC-H price/performance metric is expressed as \$/QphH@Size.

TPC-DS (Poess et al. 2002; Poess et al. 2007; Nambiar and Poess 2006; Poess et al. 2017) is an enterprise-class benchmark, published and maintained by the Transaction Processing Performance Council (TPC), to measure the performance of decision support systems running on SQL-based big data systems. TPC-DS models several generally applicable aspects of an SQL-based big data system, including queries and data maintenance. The benchmark provides a representative evaluation of performance as a general-purpose decision support system. A benchmark result measures query response time in single-user mode, query throughput in multiuser mode, and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex decision support workload. The purpose of TPC-DS benchmarks is to provide relevant, objective performance data to industry users.

TPC-DI (Poess et al. 2014) is an enterprise-class data integration benchmark, modeled after an online brokerage firm. Historically, the process of synchronizing a decision support system with data from operational systems has been referred to as extract, transform, load (ETL), and the tools supporting such process have been referred to as ETL tools. Recently, ETL was replaced by the more comprehensive acronym, data integration (DI). DI describes the process of extracting and combining data from a variety of data source formats, transforming that data into a unified data model representation and loading it into a data store. The TPC-DI benchmark combines and transforms data extracted from an online transaction processing (OTLP) system along with other sources of data and loads it into a data warehouse. The source and destination data models, data transformations, and implementation rules have been designed to be broadly representative of modern data integration requirements.

TPC-VMS is TPC's first benchmark to measure the performance of database workloads run in virtualized environments. Introduced in 2012, the TPC Virtual Measurement Single System Specification (TPC-VMS) leverages the TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks by adding the methodology and requirements for running and reporting performance metrics for virtualized databases. The intent of TPC-VMS is to represent a virtualization environment where three database workloads are consolidated onto one server. Test sponsors choose one of the four benchmark workloads (TPC-C, TPC-E, TPC-H, or TPC-DS) and run one instance of that benchmark workload in each of the three virtual machines (VMs) on the system under test. The three virtualized databases must have the same attributes, e.g., the same number of TPC-C warehouses, the same number of TPC-E load units, or the same TPC-DS or TPC-H scale factors. The TPC-VMS Primary Performance Metric is the minimum value of the three TPC Benchmark Primary metrics for the TPC benchmarks run in the virtualization environment.

TPCx-V measures the performance of a virtualized server platform under a demanding database workload. It stresses CPU and memory hardware, storage, networking, hypervisor, and the guest operating system. TPCx-V workload is database-centric and models many properties of cloud services, such as multiple VMs running at different load demand levels, and large fluctuations in the load level of each VM. Unlike previous TPC benchmarks, TPCx-V has a publicly available, end-to-end benchmarking kit, which was developed specifically for this benchmark. It loads the databases, runs the benchmark, validates the results, and even performs many of the routine audit steps. Another unique characteristic of TPCx-V is an elastic workload that varies the load delivered to each of the VMs by as much as 16x while maintaining a constant load at the host level.

TPCx-BB measures the performance of Hadoop-based big data systems. It measures the performance of both hardware and software

components by executing 30 frequently performed analytical queries in the context of retailers with physical and online store presence. The queries are expressed in SQL for structured data and in machine learning algorithms for semi-structured and unstructured data. The SQL queries can use Hive or Spark, while the machine learning algorithms use machine learning libraries, user-defined functions, and procedural programs.

TPCx-HCI benchmark measures the performance of hyper-converged infrastructure clusters under a demanding database workload. It stresses the virtualized hardware and software of converged storage, networking, and compute resources of the HCI platform. The TPCx-HCI benchmark is based on the TPCx-V benchmark and uses the same kit as TPCx-V, but the results of these benchmarks are not comparable. The number of VMs is calculated differently for the two benchmarks, even on similar hardware. The TPCx-HCI workload is database-centric and models many properties of cloud services, such as multiple VMs running at different load demand levels, and large fluctuations in the load level of each VM. The TPCx-HCI benchmarking kit, which was developed from scratch, is a complete end-to-end kit that loads the databases, runs the benchmark, validates the results, and even performs many of the routine audit steps. Two unique characteristics of TPCx-HCI are:

1. It has an elastic workload that varies the load delivered to each of the VMs by as much as 16x while maintaining a constant load at the cluster level. Sustaining optimal throughput for this elastic workload on a multi-node HCI cluster would typically benefit from frequent migrations of VMs to rebalance the load across nodes. This property measures the efficiency of VM migration as well as the uniformity of access to data from all the nodes.
2. In the data accessibility test of TPCx-HCI, a node is powered down ungracefully, and the benchmark continues to run on the other nodes. The test sponsor has to include a

throughput graph for this test, demonstrating the impact on performance, as well as report the recovery time to regain resilience.

TPCx-IoT is the industry's first benchmark that enables direct comparison of different software and hardware solutions for gateway systems in the Internet of Things (IoT). Positioned between edge architecture and the back-end data center, gateway systems perform functions such as data aggregation, real-time analytics, and persistent storage. TPCx-IoT was specifically designed to provide verifiable performance, price/performance, and availability metrics for commercially available systems that typically ingest massive amounts of data from large numbers of devices while running real-time analytic queries. The workload is representative of activities typical in IoT gateway systems, running on commercially available hardware and software platforms. The TPCx-IoT can be used to assess a broad range of system topologies and implementation methodologies in a technically rigorous and directly comparable, vendor-neutral manner.

The TPC-Pricing Specification and TPC-Energy Specification are common across all the benchmark standards. **TPC Pricing** is a consistent set of pricing procedures and methodologies for all TPC benchmarks. TPC Pricing includes various pricing methodologies and models to allow benchmark sponsors to produce verifiable prices for benchmarks executed on both physically acquired hardware and licensed compute services (Cloud). **TPC-Energy** standard (TPC, 16) is designed to augment existing TPC benchmarks with energy metrics, so that end users can understand the energy costs associated with a specific benchmark result.

TPC Foundations

Before the release of any new benchmark standard, the TPC creates a lengthy and detailed definition of the new benchmark. The resulting specifications are dense documents with stringent

requirements. These very complete specifications help ensure that all published benchmark results are comparable. TPC members also constantly work to update and improve specifications to help them stay current and complete.

Unique to the TPC is the requirement that all published benchmarks be audited by an independent third party, which has been certified by the organization. This requirement ensures that published results adhere to all of the very specific benchmark requirements and that results are accurate so any comparison across vendors or systems is, in fact, comparing "apples to apples."

The end result is that the TPC creates benchmark standards that reflect typical database workloads. The process of producing a benchmark is highly structured and audited so that valid assessments can be made across systems and vendors for any given benchmark. Reported results include performance, price/performance, and energy/performance, which help customers identify systems that deliver the highest level of performance, using the least amount of energy.

Key Applications

TPC benchmarks are designed to measure the performance and price/performance of a broad range of application environments. TPC benchmarks use performance methodologies in a technically rigorous and directly comparable, vendor-neutral manner both on-premise and in the cloud.

Cross-References

- ▶ [Big Data Benchmark](#)
- ▶ [Cloud Big Data Benchmarks](#)
- ▶ [End-to-end Benchmark](#)
- ▶ [Energy Benchmarking](#)
- ▶ [Metrics for Big Data Benchmarks](#)

References

- Hogan T (2009) Overview of TPC benchmark E: the next generation of OLTP benchmarks. In: Nambiar R, Poess M (eds) Performance evaluation and benchmarking: LNCS, vol 5895. Springer ISBN 978-3-642-10423
- Nambiar R, Poess M (2006) The making of TPC-DS. VLDB. pp 1049–1058
- Poess M, Floyd C (2000) New TPC benchmarks for decision support and web commerce. SIGMOD Rec 29(4):64–71
- Poess M, Smith B, Kollár L, Larson PÅ (2002) TPC-DS, taking decision support benchmarking to the next level. In: SIGMOD conference 2002, pp 582–587
- Poess M, Nambiar RO, Walrath D (2007) Why you should run TPC-DS: a workload analysis. VLDB. pp 1138–1149
- Poess M, Rabl T, Jacobsen H-A (2017) Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. SoCC. pp 573–585
- Poess M, Nambiar R, Kulkarni K, NarasimhaDevaray C, Rabl T, Jacobsen H-A (2018) Analysis of TPCx-IoT: the first industry standard benchmark for IoT gateway systems. ICDE
- Poess M, Rabl T, Jacobsen H-A, Caufield B (2014) TPC-DI: the first industry benchmark for data integration. PVLDB 7(13):1367–1378
- Pöss M, Floyd C (2000) New TPC benchmarks for decision support and web commerce. SIGMOD Rec 29(4):64–71
- Raab F (1993) TPC-C – the standard benchmark for online transaction processing (OLTP). In: The benchmark handbook
- TPC Transaction processing performance council homepage, in <http://www.tpc.org>
- TPC TPC policies in http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp

TPC-DS

Meikel Poess
Server Technologies, Oracle Corporation,
Redwood Shores, CA, USA

Synonyms

[Big data benchmark](#)

Definitions

TPC-DS is an enterprise-class benchmark, published and maintained by the Transaction Processing Performance Council (TPC), to measure the performance of decision support systems running on SQL-based big data systems. TPC-DS models several generally applicable aspects of an SQL-based big data system, including queries and data maintenance. The benchmark provides a representative evaluation of performance as a general-purpose decision support system. A benchmark result measures query response time in single-user mode, query throughput in multiuser mode, and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex decision support workload. The purpose of TPC-DS benchmarks is to provide relevant, objective performance data to industry users.

TPC Big Data Sort Benchmark

► [TPCx-HS](#)

TPC Express Benchmark HS

► [TPCx-HS](#)

Historical Background

TPC's decision to split the benchmark TPC-D into two new benchmarks, TPC-H and TPC-R, in February 26, 1999, was meant to be a temporary solution, until a more modern decision support benchmark was developed. It took the TPC 8 years to develop a new benchmark and 5 additional years to finally release the first version of TPC-DS on January 17, 2012. The second version of TPC-DS was published on August 28, 2015. The second version of TPC-DS was specifically designed to measure the performance of decision support systems (DSS) running on

SQL-based big data systems. The most recent version of TPC-DS can be accessed at [http://www\(tpc.org/%20TPC_Documents_Current_Versions/%20pdf/TPC-DS_v2.8.0.pdf](http://www(tpc.org/%20TPC_Documents_Current_Versions/%20pdf/TPC-DS_v2.8.0.pdf).

Foundations

Business Environment

TPC-DS models any industry that must manage, sell, and distribute products, for instance, food, electronics, furniture, music and toys, etc. It utilizes the business model of a large retail company having multiple stores located nationwide. Beyond its brick and mortar stores, the company also sells goods through catalogs and the Internet. Along with modeling a system to manage sales and returns, it models the data of a simple inventory system and a promotion system. The following are examples of business processes of this retail company:

- Record customer purchases and track customer returns from any of the three sales channels.
- Find revenue impact of price changes based on promotions.
- Maintain warehouse inventory.
- Create dynamic web pages.

TPC-DS does not benchmark the operational side of the business. It is assumed that the sales channel subsystems were designed at different times by diverse groups having dissimilar functional requirements. It is also recognized that they may be operating on significantly different hardware configurations, software configurations, and data model semantics. All three sales channel subsystems are autonomous and retain possibly redundant information regarding customer profiles and their addresses.

TPC-DS' modeling of the business environment falls into three broad categories:

- Data model
- Query and user model
- Data maintenance model

Data Model

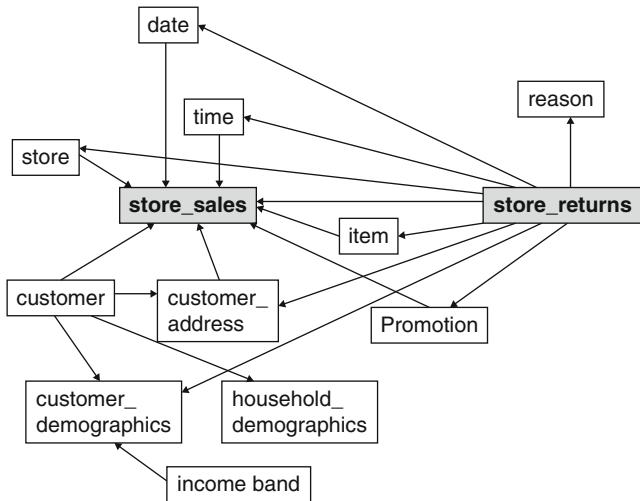
TPC-DS models a system that allows potentially long-running and multipart queries where the DBA can assume that the data processing system is quiescent for queries during any particular period. TPC-DS' data tracks, possibly with some delay, the state of an operational database through data maintenance functions, which include a number of modifications impacting some part of the decision support system. The schema is a snowflake schema consisting of multiple dimension and fact tables. Each dimension table has a single-column surrogate key. The fact tables join dimensions using each dimension table's surrogate key. The dimension tables can be classified into one of the following types:

- Static, which means that the contents of the dimension are loaded once during database load and do not change over time. The date dimension is an example of a static dimension.
- Historical, which means that the history of the changes made to the dimension data is maintained by creating multiple rows for a single business key value. Each row includes columns indicating the time period for which the row is valid. The fact tables are linked to the dimension values that were active at the time the fact was recorded, thus maintaining "historical truth." Item is an example of a historical dimension.
- Nonhistorical, which means that the history of the changes made to the dimension data is not maintained. As dimension rows are updated, the previous values are overwritten, and this information is lost. All fact data is associated with the most current value of the dimension. Customer is an example of a nonhistorical dimension.

Schema

The TPC-DS schema models the sales and returns for an organization that employs three primary sales channels: *stores*, *catalogs*, and the *web*. The relationship between tables of the store channel tables is given in Fig. 1. The schema, organized as a snowflake schema, includes seven fact tables,

TPC-DS, Fig. 1
Store_sales and
store_returns relationships



a pair of fact tables that models product sales and returns for each of the three channels, and one single fact table that models warehouse inventory for the catalog and Internet sales channels. In addition, the schema includes 17 dimension tables that are associated with all sales channels. As an extension to the pure star schema, the snowflake schema separates static data in the outlying dimension tables from the more dynamic data in the inner dimension tables and the fact tables. That is, in addition to their relation to the fact table, dimensions can have relations to other dimensions.

Dataset Scaling

TPC-DS deploys a data scaling approach that is a hybrid between domain and dataset scaling. Domain scaling keeps the number of tuples fixed but expands the domains used to generate them. For example, there could be a new type of store introduced in a retail dataset, or it could cover a longer historical period. Dataset scaling expands the number of tuples in a dataset but keeps the underlying value sets (the domains) static. The business analogy for dataset scaling would be a system where the number of items remains static but the volume of transactions per year increases. Most table columns in TPC-DS employ dataset scaling instead of domain scaling, especially fact table columns. Some columns in small tables employ domain scaling. The do-

mains have to be scaled down to adjust for the lower table cardinality. For instance, the domain for county is approximately 1800. At scale factor 100, there exist only about 200 stores. Hence, the county domain had to be scaled down for stores.

Fact tables scale linearly with the scale factor, while dimension tables scale sublinearly. Consequently, the unrealistic scaling issues that TPC-H is encountering are avoided. Even at larger-scale factors, the TPC-DS number of customers, items, etc. remains realistic. At scale factor 100, the database contains approximately 100GB of raw data. That is, 58 million items are sold per year by 2 million customers in 200 stores. On average each shopping card contains 10.5 items.

T

Workload

TPC-DS models user queries and data maintenance operations. The queries convert operational facts into business intelligence, while the data maintenance operations synchronize the operational side of a business with the data warehouse. Taken together, the combination of these two workloads constitutes the core competencies of any decision support system. The TPC has carefully evaluated the TPC-DS workload to provide a robust, rigorous, and complete means for the evaluation of systems meant to provide that

competency. The queries modeled by the benchmark exhibit the following characteristics:

- They address complex business problems.
- They use a variety of access patterns, query phrasings, operators, and answer set constraints.
- They employ query parameters that change across query executions.

In order to address the enormous range of query types and user behaviors encountered by a decision support system, TPC-DS utilizes a generalized query model to generate the queries. This model allows the benchmark to capture important aspects of the interactive, iterative nature of online analytical processing (OLAP) queries, the longer-running complex queries of data mining and knowledge discovery, and the more planned behavior of well-known report queries.

The size of the schema and its three sales channels allow for amalgamating the above query classes, especially ad hoc and reporting, into the same benchmark. An ad hoc querying workload simulates an environment in which users connected to the system send individual queries that are not known in advance. The system's administrator (DBA) cannot optimize the system specifically for this set of queries. Consequently, execution time for those queries can be very long. In contrast, queries in a reporting workload are very well known in advance. As a result, the DBA can optimize the system specifically for these queries to execute them very rapidly by using clever data placement methods (e.g., partitioning and clustering) and auxiliary data structures (e.g., materialized views and indexes). Amalgamating both types of queries has been traditionally difficult in benchmark environments since per the definition of a benchmark, all queries, apart from bind variables, are known in advance. TPC-DS accomplishes this fusion by dividing the schema into reporting and ad hoc parts. The catalog sales channel is dedicated for the reporting part, while the store and web channels are dedicated for the ad hoc part. The catalog sales channel was chosen as the reporting part because its data accounts for

40% of the entire dataset. The reporting and ad hoc parts of the schema differ in what kind of auxiliary data structures can be created. The idea behind this approach is that the queries accessing the ad hoc part constitute the ad hoc query set, while the queries accessing the reporting part are considered the reporting queries.

A sophisticated decision support system must support a diverse user population. While there are many ways to categorize those diverse users and the queries that they generate, TPC-DS has defined four broad classes of queries that characterize most decision support queries:

- Reporting queries
- Ad hoc queries
- Iterative OLAP queries
- Data mining queries

Reporting queries capture the “reporting” nature of a DSS system. They include queries that are executed periodically to answer well-known, predefined questions about the financial and operational health of a business. Although reporting queries tend to be static, minor changes are common. From one use of a given reporting query to the next, a user might choose to shift focus by varying a date range, a geographic location, or a brand name.

Ad hoc queries capture the dynamic nature of a DSS system in which impromptu queries are constructed to answer immediate and specific business questions. The central difference between ad hoc queries and reporting queries is the limited degree of foreknowledge that is available to the system administrator when planning for an ad hoc query.

Iterative OLAP queries allow for the exploration and analysis of business data to discover new and meaningful relationships and trends. While this class of queries is similar to the “ad hoc queries” class, it is distinguished by a scenario-based user session in which a sequence of queries is submitted. Such a sequence may include both complex and simple queries.

Data mining is the process of sifting through large amounts of data to produce data content

relationships. It can predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions. This class of queries typically consists of joins and large aggregations that return large data result sets for possible extraction.

Implementation Rules

A TPC-DS database must be implemented using commercially available data processing software, and its queries must be executed via an SQL interface. More generally, the implementation must be based on systems, products, technologies, and pricing that:

- Are generally available to users
- Are relevant to the market segment that the individual TPC benchmark models or represents (e.g., TPC-DS models and represents complex, high data volume, decision support environments)
- Would plausibly be implemented by a significant number of users in the market segment modeled or represented by the benchmark

The use of new systems, products, technologies (hardware or software), and pricing is encouraged so long as they meet the requirements above. Specifically prohibited are benchmark systems, products, technologies, or pricing (hereafter referred to as “implementations”) whose primary purpose is performance optimization of TPC benchmark results without any corresponding applicability to real-world applications and environments. In other words, all “benchmark special” implementations, which improve benchmark results, but are not real-world performance or pricing, are prohibited.

The use of explicitly created auxiliary data structures (EADS) is highly regulated in TPC-DS. In case their use is allowed, all replicated data must be managed by the system used for the benchmark implementation, all replications must be transparent to all data manipulation operations, their creation must be included in the database load test, and they may not be created or deleted during the performance test.

Auxiliary data structures that do not include data materialized from Catalog_Sales or Catalog_Returns are subject to the following limitations:

- They may materialize data from no more than one base table.
- They may materialize all or some of the following three items:
 1. The primary key or any subset of PK columns if the PK is a compound key
 2. Pointers or references to corresponding base table rows (e.g., “row IDs”)
 3. At most one of the following:
 - A foreign key or any subset of the FK columns if the FK is a compound key
 - A column having a date data type
 - A column that is a business key

Auxiliary data structures that include data materialized from Catalog_Sales or Catalog_Returns may not include any data materialized from Store_Sales, Store_Returns, Web_Sales, Web_Returns, or Inventory. Auxiliary data structures that materialize data from both fact and dimension tables must be the result of joins between FK- and PK-related columns. Auxiliary data structures that materialize data from one or more dimension tables without simultaneously materializing data from the primary key or any subset of PK columns if the PK is a compound key. Auxiliary data structures that materialize data from one or more dimension tables must materialize at least one dimension row for every fact table row, unless the foreign key value for a dimension row is null.

T

Partitioning Schemas

Horizontal partitioning of base tables or EADS is allowed. If the partitioning is a function of data in the table or EADS, the assignment of data to partitions must be based on the values in the partitioning column(s). Only primary keys, foreign keys, date columns, and date surrogate keys may be used as partitioning columns. If partitioning DDL uses directives that specify explicit partition

values for the partitioning columns, they must satisfy the following conditions:

- They may not rely on any knowledge of the data stored in the partitioning column(s) except the minimum and maximum values for those columns and the definition of data types for those columns.
- Within the limitations of integer division, they shall define each partition to accept an equal portion of the range between the minimum and maximum values of the partitioning column(s).
- For date-based partitions, it is permissible to partition into equally sized domains based upon an integer granularity of days, weeks, months, or years, all using the Gregorian calendar (e.g., 30 days, 4 weeks, 1 month, 1 year, etc.). For date-based partition granularities other than days, a partition boundary may extend beyond the minimum or maximum boundaries as established in that table's data characteristics.
- The directives shall allow the insertion of values of the partitioning column(s) outside the range covered by the minimum and maximum values.
- Multilevel partitioning of base tables or auxiliary data structures is allowed only if each level of partitioning satisfies the general partition conditions as stated above.
- Vertical partitioning of base tables or EADS is allowed when meeting all of the following requirements:
 - SQL DDL that explicitly partitions data vertically is prohibited.
 - SQL DDL must not contain partitioning directives which influence the physical placement of data on durable media.
 - The row must be logically presented as an atomic set of columns.

Constraints

The use of both enforced and unenforced constraints is permitted. If constraints are used, they must satisfy the following requirements:

- Enforced constraints shall be enforced either at the statement level or at the transaction level.
- Unenforced constraints must be validated after all data is loaded during the load test and before the start of the performance test.
- They are limited to primary key, foreign key, and NOT NULL constraints.

NOT NULL constraints are allowed on EADS and tables. Only columns that are marked “N” in their logical table definition (or columns in EADS derived from such columns) can be constrained with NOT NULL. If foreign key constraints are defined and enforced, there is no specific requirement for a particular delete/update action when enforcing a constraint (e.g., ANSI SQL RESTRICT, CASCADE, and NO ACTION are all acceptable).

The Data Accessibility Properties

The system under test (SUT) must be configured to satisfy the requirements for data accessibility. Data accessibility is demonstrated by the SUT being able to maintain operations with full data access during and after the permanent irrecoverable failure of any single durable medium containing tables, EADS, or metadata. Data accessibility tests are conducted by inducing failure of a durable medium within the SUT. The medium to be failed is to be chosen at random by the auditor.

The data accessibility test is performed by causing the failure of a single durable medium during the execution of the first data maintenance test. The data accessibility test is successful if all in-flight and subsequent queries and data maintenance functions complete successfully. The data accessibility test must be performed as part of the performance test that is used as the basis for reporting the performance metric and results while running against the test database at the full reported scale factor.

Execution Rules

A full run of TPC-DS consists of the following tests:

1. Database Load Test – the database load test is the process of building the test database.
2. Power Test – the power test is executed immediately following the load test. It measures the ability of the system to process a sequence of queries in the least amount of time in a single-user fashion. The queries in the power test shall be executed in the order assigned to its stream identification number:
3. Throughput Test 1 – the throughput tests measure the ability of the system to process the most queries in the least amount of time with multiple users. It immediately follows the power test.
4. Data Maintenance Test 1 – the data maintenance tests measure the ability to perform desired data changes to the TPC-DS dataset. Each data maintenance test executes $S_q/2$ refresh runs, with S_q being the number of query streams. Each refresh run uses its own dataset as generated by dsdgen. Refresh runs must be executed in the order generated by dsdgen. Refresh runs do not overlap; at most one refresh run is running at any time. All data maintenance functions need to have finished in refresh run n before any data maintenance function can commence on refresh run $n + 1$.
5. Throughput Test 2 – repeat of the first throughput test.
6. Data Maintenance Test 2 – repeat of the first data maintenance test.

Primary Performance Metric

TPC-DS defines three primary metrics:

1. Performance metric, QphDS@SF, reflecting the TPC-DS query throughput, defined as follows:

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{LD}}} \right\rfloor$$

where:

- SF is defined in Clause 3.1.3 and is based on the scale factor used in the benchmark.

- Q is the total number of weighted queries: $Q = S_q * 99$, with S_q being the number of streams executed in a throughput test.
- $T_{PT} = T_{Power} * S_q$, where T_{Power} is the total elapsed time to complete the power test and S_q is the number of streams executed in a throughput test.
- $T_{TT} = T_{TT1} + T_{TT2}$, where T_{TT1} is the total elapsed time of throughput test 1 and T_{TT2} is the total elapsed time of throughput test 2.
- $T_{DM} = T_{DM1} + T_{DM2}$, where T_{DM1} is the total elapsed time of data maintenance test 1 and T_{DM2} is the total elapsed time of data maintenance test 2.
- T_{LD} is the load factor computed as $T_{LD} = 0.01 * S_q * T_{Load}$, where S_q is the number of streams executed in a throughput test and T_{Load} is the time to finish the load.
- T_{PT} , T_{TT} , T_{DM} , and T_{LD} quantities are in units of decimal hours with a resolution of at least 1/3600th of an hour (i.e., 1 s).
- 2. Price-performance metric, $$/QphDS@SF$, as defined as follows:

$$$/QphDS@SF = \frac{P}{QphDS@SF}$$

where:

- P is the price of the priced system.
- QphDS@SF is the reported performance metric.
- 3. System availability date.

T

Cross-References

- [Big Data Architectures](#)
- [Big Data Benchmark](#)
- [Cloud Big Data Benchmarks](#)
- [End-to-end Benchmark](#)
- [Energy Benchmarking](#)
- [Metrics for Big Data Benchmarks](#)
- [TPC](#)
- [TPC-H](#)

References

- Barata M, Bernardino J, Furtado P (2015) An overview of decision support benchmarks: TPC-DS, TPC-H and SSB. *WorldCIST* 1:619–628
- Chen D, Ye X, Wang J (2013) A multidimensional data model for TPC-DS benchmarking. *Internetware*, pp 21:1–21:4
- Current TPC-DS specification http://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v2.6.0.pdf
- Nambiar RO, Poess M (2006) The making of TPC-DS. VLDB, pp 1049–1058
- Poess M, Rabl T, Jacobsen H-J (2017) Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. *SoCC*, pp 573–585
- Pöss M, Smith B, Kollár L, Larson P-Å (2002) TPC-DS, taking decision support benchmarking to the next level. *SIGMOD conference*, pp 582–587
- Pöss M, Nambiar RO, Walrath D (2007) Why you should run TPC-DS: a workload analysis. VLDB, pp 1138–1149
- Shi L, Huang B, Xu H, Ye X (2010) Implementation of TPC-DS testing tool. DBTA, pp 1–4
- Zhao H, Ye X (2013) A practice of TPC-DS multidimensional implementation on NoSQL database systems. TPCTC, pp 93–108

TPC-H

Meikel Poess
Server Technologies, Oracle Corporation,
Redwood Shores, CA, USA

Synonyms

[Ad hoc benchmark](#); [Data warehouse benchmark](#)

Definitions

An enterprise class decision support benchmark by the Transaction Processing Performance Council examines large volumes of data to answer critical ad hoc business questions. It executes a suite of complex ad hoc SQL queries

in single- and multiuser modes together with concurrent data modifications. The queries and the data populating the database have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation.

Historical Background

The first decision support benchmark, TPC-D, was released to the public by the Transaction Processing Performance Council (TPC) on May 5, 1995. For the technology available at that time, TPC-D imposed many challenges both on hardware and on DBMS systems. It implemented a data warehouse using a 3rd normal form (3NF) schema consisting of eight tables. Anticipating the data explosion of data warehouses in the industry, TPC-D was developed to scale from 1 GB to 3 TB of raw data pushing IO subsystems to their limit.

Toward the end of the 1990s, DBMS vendors developed aggregate/summary structures, commonly referred to as join indices, summary tables, and materialized views that are automatically maintained and transparently used by the query optimizer via query rewrite. These types of auxiliary data structures are typically used in reporting benchmarks to speed up queries that are well known in advance. As the cost of creating and maintaining these auxiliary data structures was not accounted for in the metric of TPC-D, this technology decreased query elapsed times considerably, resulting in an over proportional increase in the main performance metric. As a result, the TPC recognized that TPC-D was effectively broken.

Unable to integrate two workloads typical for decision support systems, ad hoc and reporting, the TPC constituted two separate benchmarks, TPC-R for a reporting workload and TPC-H for an ad hoc querying workload. An ad hoc querying workload simulates an environment in which users connected to the database system send individual queries that are not known in advance. Hence, the system's database administrator (DBA) cannot optimize the database system specifically for these queries. Consequently, exe-

cution time for these queries can be very long. On the other hand, queries in a reporting workload are very well known in advance, because they are executed periodically, for example, daily sales reports. As a result, the DBA can optimize the database system specifically for these queries to execute them very rapidly.

TPC-R was never adopted by the industry and was decommissioned by the TPC in the late 2002. TPC-H, on the other hand, continued to thrive. As of November 2017, there have been 329 TPC-H benchmark results published.

Foundations

Business Environment

The use case of TPC-H is built on any industry that has to manage, sell, and distribute products or services worldwide, similar to a car rental company, food distributor, and parts manufacturer. The use case is divided into two large areas, a *business operation* area and a *decision support* area. The *business operation* models the operational end of the business environment where transactions are executed on a real-time basis. Performance of systems managing this part is represented in other TPC benchmarks such as TPC-C and TPC-E and is not modeled in TPC-H. TPC-H focuses on executing the queries of the *decision support* part of a business where trends are computed and refined data are produced to support the making of sound business decisions. These queries:

- Give answers to real-world business questions
- Simulate ad hoc queries, for instance, issued via a point and click GUI interface
- Are far more complex than most OLTP transactions
- Include a rich breadth of operators and selectivity constraints
- Generate intensive activity on the part of the database server component of the system under test
- Are executed against a database complying to specific population and scaling requirements

- Are implemented with constraints derived from staying closely synchronized with an online production database

The TPC-H operations are modeled as follows:

- The database is continuously available 24 h a day, 7 days a week, for ad hoc queries from multiple end users and data modifications against all tables, except possibly during infrequent, for instance, once a month, maintenance sessions.
- The TPC-H database tracks, possibly with some delay, the state of the OLTP database through ongoing refresh functions, which batch together a number of modifications impacting some part of the decision support database.
- Due to the worldwide nature of the business data stored in the TPC-H database, the queries and the refresh functions may be executed against the database at any time, especially in relation to each other. In addition, this mix of queries and refresh functions is subject to specific ACID requirements, since queries and refresh functions may execute concurrently.
- To achieve the optimal compromise between performance and operational requirements, the database administrator can set, once and for all, the locking levels and the concurrent scheduling rules for queries and refresh functions.

T

Business Environment

TPC-H uses the same 3rd normal form schema as its predecessor TPC-D did. It consists of eight base tables, *lineitem*, *orders*, *part*, *partsupp*, *customer*, *supplier*, *nation*, and *region*. TPC supplies a data generator (dbgen) that generates data for all base tables depending on a scale factor (SF). The scale factor determines the size of the raw as generated by dbgen, for instance, SF = 100 means that the sum of all base tables equals about 100 GB. While dbgen accepts any integer between 1 and 100,000, scale factors that are valid for publication are

limited to 10, 30, 100, 300, 1000, 3000, 10,000, 30,000, and 100,000. Scale factor 1 is not valid for publication but is used to generate a small database that is used for demonstrating ACID and answer set correctness. The number of rows in all tables, except for nation and region, scales proportionally with the scale factor. *Lineitem* and *orders* are the two largest tables. They contain about 83% of the data.

Workload

The workloads in TPC-H consist of a database load, the execution of 22 queries in both single and multiuser mode, and two refresh operations. The process of building the test database is known as “database load.” The database load time includes all of the elapsed time to create the tables, load data, create optional indices, define and validate optional constraints, gather statistics configure the system, and ensure that the test database meets the ACID requirements including syncing loaded data on RAID devices and the taking of a backup of the database, when necessary.

Two refresh operations (RF1, RF2) model the loading of new sales information (RF1) and the purging of stale or obsolete sales information (RF2) from the database. RF1 inserts new rows into the tables *lineitem* and *orders*, while RF2 removes the same amount of rows from those tables.

Four components, a business question, a functional query definition, substitution parameters, and a query validation, characterize the 22 queries. The business question illustrates the business context in which the query is used. The functional query definition defines the function to be performed by the query in SQL-92 language (ISO/IEC 9075:1992). Each query is defined as a query template. Before a query can be executed against a database, parameters are substituted with values generated by the supplied program *qgen* in such way that the performance for different queries is comparable. The query validation describes how to validate each query by executing them against a 1 GB database and comparing their results against a predefined result set.

The 22 queries are run in single-user mode, which is in one database session, to demonstrate the ability of the test system to use all of the resources for a single-user and in multiuser mode, which is in multiple concurrent database sessions, to demonstrate the ability of the systems to use all of the resources to satisfy concurrent users. The queries are intended to test most query capabilities in respect to a typical decision support system. They:

Implementation Rules

The database must be implemented using a commercially available database management system (DBMS). Physical clustering of records within the database is allowed as long as this clustering does not alter the logical independence of each table. The intent of this clause is to permit flexibility in the physical design of a database while preserving a strict logical view of all the tables.

Partitioning Schemes

Horizontal partitioning of base tables or auxiliary structures created by database directives is allowed. If this assignment is a function of data in the table or auxiliary structure, the assignment must be based on the value of a partitioning field. A partitioning field must be one and only one of the following:

- A primary key of the table, whether or not it is defined as a primary key constraint
- A foreign key of the table, whether or not it is defined as a foreign key constraint
- A column having a date datatype

If directives are used that specify explicit values for the partitioning field, they must satisfy the following conditions:

- They may not rely on any knowledge of the data stored in the table except the minimum and maximum values of columns used for the partitioning field as defined in the specification.
- Within the limitations of integer division, they must define each partition to accept an equal

portion of their range between the minimum and maximum values of the partitioning column(s). For date-based partitions, it is permissible to partition into equally sized domains based upon an integer granularity of days, weeks, months, or years (e.g., 30 days, 4 weeks, 1 month, 1 year, etc.). For date-based partition granularities other than days, a partition boundary may extend beyond the minimum or maximum boundaries.

- The directives must allow the insertion of values of the partitioning column(s) outside the range covered by the minimum and maximum values.
- Multiple-level partitioning of base tables or auxiliary structures is allowed only if each level of partitioning satisfies the conditions stated above and each level references only one partitioning field as defined above.

Auxiliary Structures

The only type of auxiliary structure allowed in TPC-H is an index. Indexes can refer to only one base table. They can refer to a primary or foreign key as defined in the TPC-H specification, a compound primary or foreign key as defined in the TPC-H specification or any single date datatype column.

Primary Performance Metric

The primary performance metric in TPC-H is the *composite* performance metric (QphH). It equally weights the contribution of the single-user power metric and the multiuser throughput metric. It expresses the number of queries the system can perform per hour.

In order to compute the composite metric for a test system at a given scale factor, one needs to run a *power test* (single-user) followed by a *throughput test* (multiuser). The results of the power and the throughput tests are then combined to calculate the composite metric.

A power test consists of the execution of the refresh operation RF1, followed by the execution of all 22 queries in single-user mode and concluded by the execution of the refresh operation RF2. During a throughput test, multiple

concurrent sessions execute all 22 queries in a predefined order. One sequence of queries in a session is also called a “query-stream” (S_1, \dots, S_n). One pair of update functions (RF1, RF2) is executed for each query-stream. All update-pairs are executed sequentially in a separate session. The sequence of update functions is also called the “update-stream.” TPC-H requires a multi-stream run with a minimum number of streams. The minimum number of streams depends on the scale factor.

Scale factor	S (Streams)
1	2
10	3
100	4
300	6
1000	7
3000	8
10,000	9

The power metric (Power@Size) is computed as the geometric mean of the elapsed times for all queries and both refresh functions obtained from the power run. The unit is queries per hour. For a given scale factor (SF), it is computed as:

Power@Size

$$= \sqrt[24]{\prod_{i=1}^{22} QI(i, 0) * \prod_{j=1}^2 RI(j, 0)} \frac{3600 * SF}{3600}$$

with:

- **QI(i,0):** elapsed time of query Q_i obtained in the power run in seconds
- **RI(j,0):** elapsed time of the refresh function RF_j obtained in the power run in seconds
- **3600:** seconds

The throughput metric (Throughput@Size) is computed as the ratio of the total number of queries executed over the length of the measurement interval of the multi-stream run. The unit is queries per hour. For a given scale factor (SF), it

is computed as:

$$\text{Throughput@Size} = \frac{S * 22 * 3600}{T_s} * SF$$

with:

- T_s : elapsed time of the multistream run
- S: stream count
- 3600: seconds
- 22: number of queries per stream

The Composite Query-Per-Hour Performance Metric (QphH) is calculated as the geometric mean of the power and throughput metrics:

$$\begin{aligned} \text{QphH@Size, QphR@Size} \\ = \sqrt{\text{Power@Size} * \text{Throughput@Size}} \end{aligned}$$

Lastly, TPC-H defines a price/performance metric as the ratio of the total system price divided by the composite metric and an availability metric, which indicates the date the system is available for purchase by the general public.

Cross-References

- ▶ [Big Data Architectures](#)
- ▶ [Decision Support Benchmarks](#)
- ▶ [End-to-End Benchmark](#)
- ▶ [TPC](#)

References

- Ballinger C (1993) TPC-D: benchmarking for decision support. In: The benchmark handbook for database and transaction systems, 2nd edn. Morgan Kaufmann. ISBN 1-55860-292-5
- Barata M, Bernardino J, Furtado P (2015) An overview of decision support benchmarks: TPC-DS, TPC-H and SSB. *WorldCIST* (1):619–628

Boncz PA, Neumann T, Erling O (2013) TPC-H analyzed: hidden messages and lessons learned from an influential benchmark. *TPCTC*:61–76

Chiba T, Onodera T (2016) Workload characterization and optimization of TPC-H queries on Apache Spark. *ISPASS*, pp 112–121

Pöss M, Floyd C (2000) New TPC benchmarks for decision support and web commerce. *SIGMOD Rec* 29(4):64–71

TPCx-HS

Tariq Magdon-Ismail
VMware Inc., Palo Alto, CA, USA

Synonyms

[TPC Big Data sort benchmark](#); [TPC Express Benchmark HS](#)

Definitions

TPCx-HS is the first industry standard Big Data benchmark for objectively measuring the performance and price/performance of Apache Hadoop and Apache Spark compatible software distributions. It stresses both the hardware and software stacks including the operating system, execution engine (MapReduce or Spark), and Hadoop Filesystem API compatible layers. TPCx-HS can be used to assess a broad range of system topologies and implementation methodologies in a technically rigorous and directly comparable, vendor-neutral manner.

Historical Background

Up until 2007, Jim Gray defined, sponsored, and administered a number of sort benchmarks (Anon et al. 1985) available to the general community. These include Minute Sort, Gray Sort, Penny Sort, Joule Sort, Datamation Sort, and TeraByte Sort. TeraByte Sort measures the amount of time

taken (in minutes) to sort 1 TB (10^{12} bytes) of data.

In 2009, Owen O’Malley and others of Yahoo! Inc. published the results of a Hadoop MapReduce implementation of TeraByte Sort called TeraSort. It was capable of sorting 100 TB of data in 173 min using 3452 nodes, each running two quad-core Intel Xeon processors with 8 GB memory and four SATA disks, breaking previously held records. Today, Big Data processing has reached mainstream status, and since its release in 2009, many companies have been making performance claims based on TeraSort that are not easily verifiable or comparable. While TeraSort nicely defines the dataset and tasks needed to measure Big Data Hadoop systems, it lacks a formal specification and enforcement of rules that enable the comparison of results across systems and vendors.

In February 2014, the *Transaction Processing Performance Council (TPC)* formed a committee to develop an industry standard benchmark based on the already popular TeraSort workload and in July that same year formally approved TPCx-HS as a standard.

Foundation

Sizing and Scale Factors

TPCx-HS follows a stepped benchmark sizing model. Unlike TeraSort, which can be scaled using an arbitrary number of rows in the dataset, TPCx-HS limits the choices to one of the following: 10 B, 30 B, 100 B, 300 B, 1000 B, 3000 B, 10000 B, 30000 B, and 100,000 B, where each row/record is 100 bytes. (There is no inherent scale limitation in the benchmark. Larger datasets can be added (and smaller ones retired) based on industry trends over time.) In TPCx-HS, these dataset sizes are referred to in terms of *scale factors*, which are defined as follows: 1 TB, 3 TB, 10 TB, 30 TB, 100 TB, 300 TB, 1000 TB, 3000 TB, and 10,000 TB, respectively. For example, a 3-terabyte scale factor corresponds to a dataset with 30-byte rows.

The primary motivation for choosing a stepped design in benchmark sizing is to ease the comparison of results across different systems. However, it should be noted that results at different scale factors are not comparable to each other due to the substantially different computational challenges found at different data volumes. Similarly, the system price/performance may not scale down linearly with a decrease in dataset size due to configuration changes required by changes in dataset size.

Data Replication

The benchmarked system, or *system under test (SUT)*, must provide data redundancy equivalent to or better than that of local JBOD storage with a replication factor of three. The alternative hardware or service must prevent data loss in the event of a permanent, irrecoverable failure of any storage device containing data generated by the benchmark modules HSGen or HSSort.

Benchmark Execution

The TPCx-HS benchmark workload consists of four modules:

- **HSGen** generates the input dataset at a particular scale factor.
- **HSSort** sorts the input dataset in total order.
- **HSValidate** validates that the output dataset is globally sorted.
- **HSDataCheck** verifies the cardinality, size, and replication factor of the dataset.

HSGen, HSSort, and HSValidate are based on TeraGen, TeraSort, and TeraValidate, respectively. The TPCx-HS kit also includes HSDataCheck, which verifies that the dataset generated by HSGen and the output produced by HSSort match the specified scale factor.

A valid benchmark run consists of five separate and non-overlapping phases that are executed sequentially. The phases are listed below:

1. Generation of input data via HSGen
2. Verification (cardinality, size, and replication) of the input data via HSDataCheck
3. Sorting the input data using HSSort

4. Verification (cardinality, size, and replication) of the sorted dataset via HSDataCheck
5. Validation of the sorted output data via HSVValidate

If any of the verification or validation phases fail, the run is considered invalid. The TPCx-HS *performance metric* for a run is based on the end-to-end run time of all five phases as illustrated in Fig. 1. To account for variance, a benchmark test consists of two identical runs – Run 1 and Run 2 – with the reported result being for the run with the lower performance metric.

No part of the SUT may be rebooted or restarted during or between the runs or any of the phases. If there is a non-recoverable error reported by any of the applications, operating system, or hardware in any of the five phases, the run is considered invalid. If a recoverable error is detected in any of the phases and is automatically dealt with or corrected by the applications, operating system, or hardware, then the run is considered valid. However, manual user intervention is not allowed. If the recoverable error requires manual intervention to deal with or correct, then the run is considered invalid. A minimum of three-way data replication must be maintained throughout the run.

The SUT cannot be reconfigured, changed, or retuned by the user during or between any of the five phases or between Run 1 and Run 2. Any manual tunings to the SUT must be performed

before the beginning of Phase 1 of Run 1 and must be fully disclosed. Any automated changes or tuning performed by the OS or commercially available product between any of the phases is allowed. Any changes to default tunings or parameters of the applications, operating systems, or hardware of the SUT must be disclosed as well.

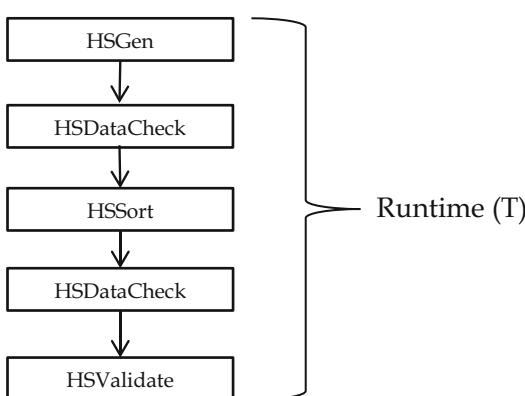
Metric

While the execution rules define the way in which a benchmark is run, the metric emphasizes the pieces that are measured (Nambiar et al. 2011). This is the most visible part of the performance data and one of the most controversial when trying to reach an agreement between different parties.

In general, a performance metric needs to be simple such that easy system comparisons are possible. If there are multiple performance metrics (e.g., A, B, C), system comparisons are difficult because vendors can claim they perform well on some of the metrics (e.g., A and C). This might still be acceptable if all components are equally important, but without this determination, there would be much debate on this issue. In order to unambiguously rank results, the TPC benchmarks focus on a single primary performance metric, which encompass all aspects of a system's performance, weighting the individual components. Taking the example from above, the performance metric M is calculated as a function of the three components A, B, and C; that is, $M = f(A, B, C)$. Consequently, TPC's performance metrics measure system and overall workload performance rather than individual component performance. In addition to the performance metric, the TPC also includes other metrics, such as price/performance metrics.

The TPC distinguishes between *primary* and *secondary* metrics. Each TPC Express Benchmark Standard (Huppler and Johnson (2013)) must define primary metrics selected to represent the workload being measured. The primary metrics must include both performance and price/performance metrics.

TPCx-HS defines three primary metrics:



TPCx-HS, Fig. 1 TPCx-HS execution phases

- HSph@SF: Composite performance metric, reflecting the TPCx-HS throughput, where SF is the scale factor
- \$/HSph@SF: Price/performance metric
- System availability date

TPCx-HS also reports the following numerical quantities:

- T_G , data generation phase completion time with HSGen reported in hh:mm:ss format
- T_S , data sort phase completion time with HSSort reported in hh:mm:ss format
- T_V , data validation phase completion time reported in hh:mm:ss format

When the TPC Energy option is chosen for reporting, the TPCx-HS energy metric reports the power/performance and is expressed as Watts/HSph@SF (see the TPCx-HS (TPCx-HS Benchmark 2018) and TPCx Energy specifications (TPC Energy 2018) for additional details and requirements).

Each secondary metric shall be referenced in conjunction with the scale factor at which it was achieved. For example, TPCx-HS TG references shall take the form of TPCx-HS TG @ SF, or “TPCx-HS TG = 2 hours @ 1.”

The primary performance metric of the benchmark is HSph@SF, the effective sort throughput of the benchmarked configuration. We use a summation method as an illustrative example:

$$\text{HSph@SF} = \left[\frac{SF}{T/3600} \right]$$

where SF is the scale factor and T is the total elapsed time for the run, in seconds.

The price/performance metric for the benchmark is defined as:

$$\$/\text{HSph@SF} = \frac{P}{\text{HSph@SF}}$$

where P is the total cost of ownership of the SUT.

The system availability date is defined in the TPC Pricing Specification (TPC Pricing 2018).

A TPCx-HS result is only comparable with other TPCx-HS results of the same scale factor. As noted before, results at the different scale factors are not comparable due to the substantially different computational challenges found at different data volumes. Similarly, the system price/performance may not scale down linearly with a decrease in dataset size due to configuration changes required by changes in dataset size.

Key Applications

TPCx-HS is designed to measure the performance and price/performance of a broad range of Hadoop and Spark system topologies and implementation methodologies, in a technically rigorous and directly comparable, vendor-neutral manner both on premise and in the cloud.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Benchmark Harness](#)
- ▶ [Big Data and Exascale Computing](#)
- ▶ [Big Data Architectures](#)
- ▶ [Big Data in the Cloud](#)
- ▶ [Cloud Big Data Benchmarks](#)
- ▶ [Computer Architecture for Big Data](#)
- ▶ [Hadoop](#)
- ▶ [Metrics for Big Data Benchmarks](#)
- ▶ [Performance Evaluation of Big Data analysis](#)
- ▶ [Scalable Architectures for Big Data analysis](#)
- ▶ [Storage Technologies for Big Data](#)
- ▶ [System Under Test](#)
- ▶ [TPC](#)
- ▶ [Virtualized Big Data Benchmarks](#)

References

Anon et al (1985) A measure of transaction processing power, a condensed version of this paper appears in Datamation, April 1, 1985. This paper was scanned from the Tandem Technical Report TR 85.2 in 2001 and reformatted by Jim Gray

- Huppler K, Johnson D (2013) TPC express – a new path for TPC benchmarks. TPCTC, pp 48–60
- Nambiar R, Wakou N, Masland A, Thawley P, Lanken M, Carman F, Majdalany M (2011) Shaping the landscape of industry standard benchmarks: contributions of the Transaction Processing Performance Council (TPC). TPCTC, pp 1–9
- TPC Energy (2018) http://www.tpc.org/tpc_energy/default.asp. Last accessed 17 Jan 2018
- TPC Pricing (2018) <http://www.tpc.org/pricing/default.asp>. Last accessed 17 Jan 2018
- TPCx-HS Benchmark (2018) <http://www.tpc.org/tpcx-hs/default.asp?version=2>. Last accessed 17 Jan 2018

Trace Clustering

Jochen De Weerdt
KU Leuven, Leuven, Belgium

Synonyms

Clustering of process instances; Process variant discovery

Definitions

Given an event log being a bag of process instances, with each process instance or trace consisting of a sequence of events, trace clustering refers to grouping these process instances so as to maximize the similarity between them and maximize the dissimilarity between the groups. While trace clustering might be described as discovering process variants, usually, the term process variant is already used for a collection of process instances with exactly the same event sequence. As such, trace clustering usually refers to the grouping of those variants into logically coherent groups, which can be, confusingly, referred to as variants of the process.

Overview

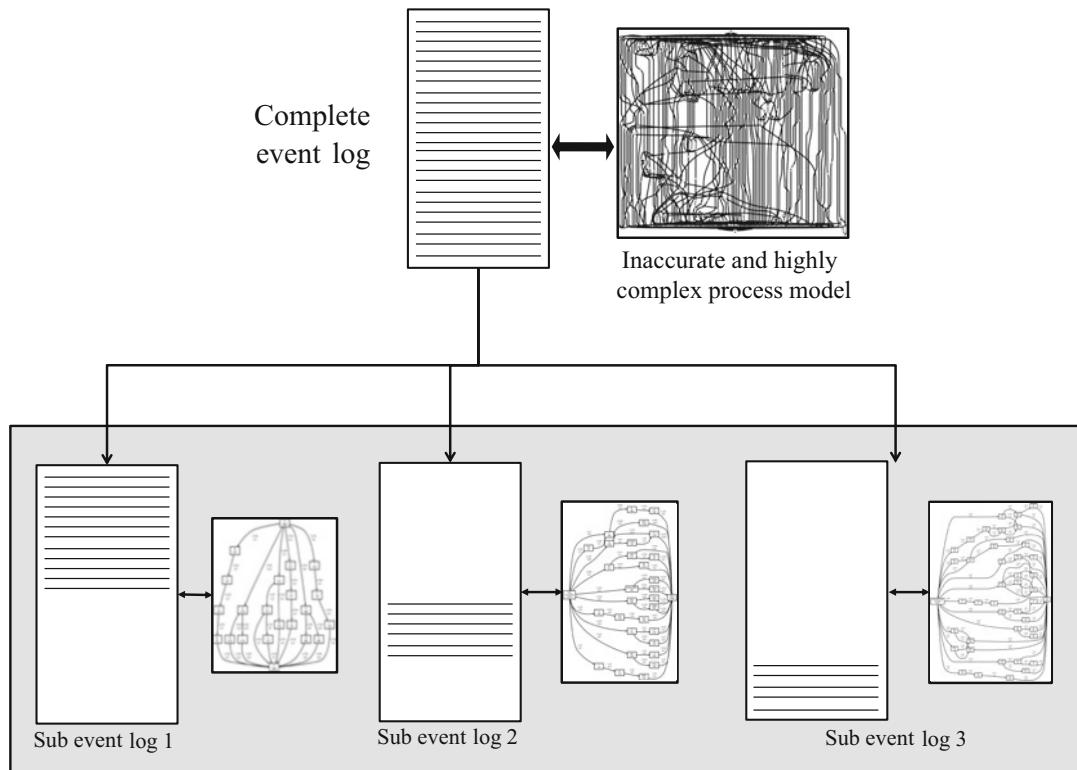
This entry discusses trace clustering in the context of process mining. The technique is

explained in general, followed by a dense yet informative typology of techniques. This entry is concluded by an overview of related topics.

Introduction

Despite the demonstrated usefulness of *automated process discovery* in flexible environments, it has been shown that its application is usually also most challenging in these contexts. More specifically, a multitude of studies illustrates that process discovery techniques experience difficulties to produce accurate and interpretable process models out of *event logs* stemming from highly flexible environments (Günther 2009; Goedertier et al. 2011; Jagadeesh Chandra Bose and van der Aalst 2009b; Veiga and Ferreira 2010). This problem is largely due to the high variety of behavior that is captured in certain event logs.

Accordingly, different approaches have been proposed to cope with this issue. Event log filtering, event log transformation (Jagadeesh Chandra Bose and van der Aalst 2009a), and tailor-made discovery techniques such as Fuzzy Mining (Günther and van der Aalst 2007) are noteworthy solutions. *Trace clustering* can be considered as another solution for reducing the complexity of the learning task at hand. This is because one can rely on multiple process models to represent the variety in behavior of a certain event log by separating execution traces into different groups. The purpose of clustering process executions is detailed in Fig. 1. By dividing traces into different groups, process discovery techniques can be applied on subsets of behavior and thus improve the accuracy and comprehensibility of discovered models. Partitioning event logs into multiple groups is a convenient approach for addressing the challenge of discovering accurate and understandable process models from logs presenting a large amount of distinct process behavior. Observe that distinct process behavior refers to the presence of a large variety of different event sequences or process variants in which the process is executed. Note that trace



Trace Clustering, Fig. 1 Creating three clusters of process instances in order to alleviate the resulting complexity of a single discovered process model

clustering does not include decomposed process discovery where traces are being split into sub-traces (e.g., corresponding to stages). As such, the difference is that in decomposed process discovery, the event log is decomposed vertically, whereas in trace clustering, it is decomposed horizontally (the traces are not split but rather grouped together).

Partitioning algorithms operating at the trace (or process instance) level are usually referred to as trace clustering techniques within the process mining domain. Observe that in some cases, there might be a natural combination of several processes into one event log. Typically, one would be able to separate out the cases pertaining to these processes based on activity names, resources, etc. However, in some cases, it might be difficult to decouple multiple processes logged into one event log, which might necessitate the use of a trace clustering technique. However, the latter can be considered as an unlikely situation, thus

not often considered as a starting point for trace clustering.

Trace Clustering Techniques

Accordingly, trace clustering is most interesting as an approach to deal with the problem that many event logs contain an extensive amount of distinct behavior (i.e., process variants), because it allows the user to split up a log so that multiple distinct models can be learnt to describe the underlying business process. Observe that, intrinsically, grouping by process variant can also be considered as trace clustering. Nevertheless, this task is trivial (and should not be subject to the use of an algorithm). As such, trace clustering techniques should operate at the level of grouping distinct process instances or process variants. Please also note that trace clustering refers to horizontal log splitting. Vertical log splitting entails

the discovery of hierarchical subprocesses, which is considered as an orthogonal learning task.

A variety of trace clustering techniques exist within the process mining field. For a recent comparative assessment, see, for instance, Thaler et al. (2015). Table 1 presents a basic overview. Trace clustering techniques often translate the problem to a propositional setting, by featurizing the process instances and subsequently applying a traditional data clustering technique. In Table 1, these techniques are referred to as “propositional.” Given that they employ distance-based data clustering algorithms, they are also considered as “distance-based.” Nevertheless, some techniques derive distances between process variants but avoid the featurization step. These techniques only have a check mark in the corresponding “distance-based” column. Finally, an entirely different set of trace clustering techniques are model-driven, i.e., they rely either on some temporal probabilistic model like a hidden Markov model, or on actual process models (in the form of Petri nets, dependency graphs, or other representations).

Propositional Trace Clustering

The simplest idea for clustering traces relies on the use of traditional data clustering techniques,

most importantly hierarchical or partitional algorithms (Jain and Dubes 1988). In order to apply these traditional data clustering techniques, one should convert an event log, i.e., a set of traces, into an attribute-value data set (i.e., the typical input format expected by data clustering techniques). As such, one should “featurize” the traces.

The first study proposing such a “featurization” approach is the pioneering study by Greco et al. (2006), who proposed a projection of the traces onto a set of suitable features, in this case a set of frequent k-grams of activity sequences. The projection of an event log onto a feature set was further explored by Song et al. (2008). Their trace clustering implementation allows for a multitude of so-called profiles to determine the vector associated with each process instance. As such, they define activity, transition, performance, case attribute profiles, etc. Furthermore, whereas Greco et al. (2006) relied on k-means, Song et al. (2008) provide an implementation that offers a full range of distance metrics and data clustering algorithms. Later, the same authors investigated the use of dimensionality reduction techniques in the context of profile-based trace clustering (Song et al. 2013). Along the same line of reasoning, Bose and van der Aalst

Trace Clustering, Table 1 An overview of trace clustering techniques and their nature

Author	Propositional	Distance-based	Model-driven
Greco et al. (2006)	✓	✓	
Song et al. (2008)	✓	✓	
Ferreira et al. (2007)		✓	
Jagadeesh Chandra Bose and van der Aalst (2009b)		✓	
Bose and van der Aalst (2010)	✓	✓	
Folino et al. (2011)			✓
De Weerdt et al. (2013)			✓
García-Bañuelos et al (2014)	(✓)	(✓)	(✓)
Delias et al. (2015)		✓	
Appice and Malerba (2015)		✓	
Evermann et al. (2016)		✓	
De Koninck and De Weerdt (2016b)			✓
Chatain et al. (2017)			✓
De Koninck et al. (2017b)			✓

(2010) proposed a refinement of the technique of Greco et al. (2006) by making use of so-called conserved patterns (e.g., maximal, super maximal, and near-super-maximal repeats) as projection features. The implementation applies hierarchical clustering instead of k-means.

Other Distance-Based Trace Clustering Techniques

In Jagadeesh Chandra Bose and van der Aalst (2009b), Bose and van der Aalst turn away from the use of an attribute-value feature space but instead regard log traces as strings over the alphabet of task labels and as such calculate the distance between traces directly by exploiting the idea of a string edit distance. However, in contrast to contemporary approaches such as the Levenshtein distance (Levenshtein 1966), specific substitution, insertion, and deletion costs are derived so as to take into account characteristic features of an event log, such as concurrency of tasks. A similar, alignment-based technique is proposed in Evermann et al. (2016). Furthermore, Delias et al. (2015) have developed a more robust spectral learning approach, where the similarity metric is adjusted in the face of significant deviating behavior. Finally, Appice and Malerba (2015) deploy a co-training strategy based on multiple views: separate clusterings are created using instance similarity based on the activities, resources, sequences, and timing aspects of the process instances and then combined to create a single multiple-view clustering.

Model-Driven Trace Clustering

A third class of trace clustering techniques can be best described as *model-driven*. Inspired by the work of Cadez et al. (2003) in the area of web usage mining, Ferreira et al. (2007) propose to cluster sequences by learning a mixture of first-order Markov models using the Expectation-Maximization (EM) algorithm. This technique has been applied in Veiga and Ferreira (2010) to server logs. Also in Folino et al. (2011), Markov chain clustering is used. Furthermore, De Weerdt et al. (2013) propose the *ActiTraC*-algorithm,

which is an active learning-inspired trace clustering technique that strives to optimize the combined fitness of the underlying process models. Finally, Chatain et al. (2017) also propose a process model-driven trace clustering technique but rely on the concept of alignments to construct models that serve as the centroids of the trace clusters.

Multi-objective Trace Clustering

Trace clustering techniques often strike a balance between multiple objectives. Typical objectives are to maximize intra-cluster trace similarity; intercluster trace dissimilarity; fitness, precision, generalization, and simplicity of the underlying process models; stability of the clustering solution; understandability by experts; and justifiability by experts (solutions need to be in line with expert knowledge). Oftentimes, trace clustering techniques inherently consider only a single optimization, e.g., propositional techniques optimize in terms of trace similarity, while only few techniques directly incorporate multiple objectives in the learning phase. In De Koninck and De Weerdt (2016b), a multi-objective trace clustering technique was presented, built on top of the active trace clustering paradigm proposed in De Weerdt et al. (2013).

Furthermore, García-Bañuelos et al. (2014) address complexity-aware trace clustering by proposing the SMD-technique, a kind of meta-level trace clustering technique which, based on a two-way divide-and-conquer discovery technique, wherein discovered process models are split, are both based on process variants and hierarchically using subprocess extraction. Observe that, because the technique can basically work on top of any other trace clustering technique, the check marks in Table 1 are put between brackets. Finally, also expert-driven trace clustering has received attention recently. In De Koninck et al. (2017b), a constraint clustering-inspired trace clustering algorithm was developed that takes both expert information (in the form of a preexisting expert clustering) and process model quality into account while grouping the traces.

Further Topics in Trace Clustering

Next to the development of trace clustering techniques, research on the topic has also addressed related problems such as the understandability of a trace clustering solution, the use of trace clustering for concept drift detection, and the determination of an appropriate number of clusters. E.g., in De Weerdt and Vanden Broucke (2014) and De Koninck et al. (2017a), a method is proposed to learn instance-level explanations of trace clusterings. Hompes et al. (2015) apply trace clustering for concept drift detection and explanation based on statistical and visual comparisons of case clusters. Finally, in De Koninck and De Weerdt (2016a, 2017), tailor-made techniques for determining the appropriate number of trace clusters have been proposed based on the concepts of stability and separation.

Cross-References

- ▶ [Automated Process Discovery](#)
- ▶ [Decomposed Process Discovery and Conformance Checking](#)
- ▶ [Event Log Cleaning for Business Process Analytics](#)

References

- Appice A, Malerba D (2015) A co-training strategy for multiple view clustering in process mining. *IEEE Trans Serv Comput* PP(99):1–1. <https://doi.org/10.1109/TSC.2015.2430327>
- Bose RPJC, van der Aalst WMP (2010) Trace clustering based on conserved patterns: towards achieving better process models. *Lect Notes Bus Inf Process* 43 LNBIP:170–181. https://doi.org/10.1007/978-3-642-12186-9_16
- Cadez I, Heckerman D, Meek C, Smyth P, White S (2003) Model-based clustering and visualization of navigation patterns on a web site. *Data Min Knowl Disc* 7(4):399–424. <https://doi.org/10.1023/A:1024992613384>
- Chatain T, Carmona J, Van Dongen B (2017) Alignment-based trace clustering. In: International conference on conceptual modeling. Springer, pp 295–308
- De Koninck P, De Weerdt J (2016a) Determining the number of trace clusters: a stability-based approach. In: van der Aalst WMP, Bergenthal R, Carmona J (eds) *Proceedings of the international workshop on algorithms & theories for the analysis of event data 2016 satellite event of the conferences: 37th international conference on application and theory of petri nets and concurrency petri nets 2016 and 16th international conference on application of concurrency to system design ACSD 2016*, Torun, 20–21 June 2016. CEUR-WS.org, CEUR Workshop Proceedings, vol 1592, pp 1–15. <http://ceur-ws.org/Vol-1592/paper01.pdf>
- De Koninck P, De Weerdt J (2016b) Multi-objective trace clustering: finding more balanced solutions. In: Dumas M, Fantinato M (eds) *Business process management workshops – BPM 2016 international workshops, Rio de Janeiro, 19 Sept 2016, Revised papers. Lecture notes in business information processing*, vol 281, pp 49–60. https://doi.org/10.1007/978-3-319-58457-7_4
- De Koninck P, De Weerdt J (2017) Similarity-based approaches for determining the number of trace clusters in process discovery. *T Petri Nets Other Models Concurr* 12:19–42. https://doi.org/10.1007/978-3-662-55862-1_2
- De Koninck P, De Weerdt J, vanden Broucke SKLM (2017a) Explaining clusterings of process instances. *Data Min Knowl Discov* 31(3):774–808. <https://doi.org/10.1007/s10618-016-0488-4>
- De Koninck P, Nelissen K, Baesens B, vanden Broucke S, Snoeck M, De Weerdt J (2017b) An approach for incorporating expert knowledge in trace clustering. In: Dubois E, Pohl K (eds) *Proceedings of the 29th international conference on Advanced information systems engineering, CAiSE 2017, Essen, 12–16 June 2017. Lecture notes in computer science*, vol 10253. Springer, pp 561–576. https://doi.org/10.1007/978-3-319-59536-8_35
- Delias P, Doumpos M, Grigoroudis E, Manolitzas P, Matsatsinis N (2015) Supporting healthcare management decisions via robust clustering of event logs. *Knowl-Based Syst* 84:203–213. <https://doi.org/10.1016/j.knosys.2015.04.012>
- De Weerdt J, Vanden Broucke S (2014) SECPI: searching for explanations for clustered process instances. In: *Lecture notes in computer science (Including subseries lecture notes artificial intelligence lecture notes in bioinformatics)*. LNCS, vol 8659, pp 408–415. https://doi.org/10.1007/978-3-319-10172-9_29
- De Weerdt J, Vanden Broucke S, Vanthienen J, Baesens B (2013) Active trace clustering for improved process discovery. *IEEE Trans Knowl Data Eng* 25(12):2708–2720. <https://doi.org/10.1109/TKDE.2013.64>
- Evermann J, Thaler T, Fettke P (2016) Clustering traces using sequence alignment. In: Reichert M, Reijers HA (eds) *Business process management workshops: BPM 2015, 13th international workshops, Innsbruck, 31 Aug–3 Sept 2015. Revised papers. Springer International Publishing*, Cham, pp 179–190. https://doi.org/10.1007/978-3-319-42887-1_15
- Ferreira DR, Zacarias M, Malheiros M, Ferreira P (2007) Approaching process mining with sequence clustering: experiments and findings. In: *BPM*, pp 360–374. https://doi.org/10.1007/978-3-540-75183-0_26

Folino F, Greco G, Guzzo A, Pontieri L (2011) Mining usage scenarios in business processes: outlier-aware discovery and run-time prediction. *Data Knowl Eng* 70(12):1005–1029. <https://doi.org/10.1016/j.data.2011.07.002>

García-Bañuelos L, Dumas M, La Rosa M, De Weerdt J, Ekanayake CC (2014) Controlled automated discovery of collections of business process models. *Inf Syst* 46:85–101

Goedertier S, De Weerdt J, Martens D, Vanthienen J, Baesens B (2011) Process discovery in event logs: an application in the telecom industry. *Appl Soft Comput* 11(2):1697–1710

Greco G, Guzzo A, Pontieri L, Saccà D (2006) Discovering expressive process models by clustering log traces. *IEEE Trans Knowl Data Eng* 18(8):1010–1027. <https://doi.org/10.1109/TKDE.2006.123>

Günther CW (2009) Process mining in flexible environments. PhD thesis, TU Eindhoven

Günther CW, van der Aalst WMP (2007) Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: ter Hofstede AHM, Benatallah B, Paik HY (eds) BPM. Lecture notes in computer science, vol 4928. Springer, pp 328–343

Hompes BFA, Buijs JCAM, van der Aalst WMP, Dixit P, Buurman J (2015) Detecting changes in process behavior using comparative case clustering. In: Ceravolo P, Rinderle-Ma S (eds) Data-driven process discovery and analysis – 5th IFIP WG 2.6 international symposium, SIMPDA 2015, Vienna, 9–11 Dec 2015, Revised selected papers. Lecture notes in business information processing, vol 244. Springer, pp 54–75. https://doi.org/10.1007/978-3-319-53435-0_3

Jagadeesh Chandra Bose RP, van der Aalst WMP (2009a) Abstractions in process mining: a taxonomy of patterns. In: Dayal U, Eder J, Koehler J, Reijers HA (eds) BPM. Lecture notes in computer science, vol 5701. Springer, pp 159–175

Jagadeesh Chandra Bose RP, van der Aalst WMP (2009b) Context aware trace clustering: towards improving process mining results. In: SDM, pp 401–412. <https://doi.org/10.1137/1.9781611972795.35>

Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Inc., Englewood Cliffs

Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions and reversals. *Sov Phys Dokl* 10:707–710

Song M, Günther CW, van der Aalst WMP (2008) Trace clustering in process mining. In: BPM workshops, pp 109–120. https://doi.org/10.1007/978-3-642-00328-8_11

Song M, Yang H, Siadat SH, Pechenizkiy M (2013) A comparative study of dimensionality reduction techniques to enhance trace clustering performances. *Expert Syst Appl* 40:3722–3737. <https://doi.org/10.1016/j.eswa.2012.12.078>

Thaler T, Ternis SF, Fettke P, Loos P (2015) A comparative analysis of process instance cluster techniques. *Wirtschaftsinformatik* 2015:423–437

Veiga GM, Ferreira DR (2010) Understanding spaghetti models with sequence clustering for prom. In: Rinderle-Ma S et al (ed) BPM workshops. LNBP, vol 43. Springer, pp 92–103. <https://doi.org/10.1007/978-3-642-12186-9>

Traces

► Spatiotemporal Data: Trajectories

Transaction Processing on Modern and Emerging Hardware

► Hardware-Assisted Transaction Processing

Transaction Processing on Persistent Memory

► Hardware-Assisted Transaction Processing: NVM

Transaction Processing on Storage-Class Memory

► Hardware-Assisted Transaction Processing: NVM

Transaction Processing Performance Council

► TPC

Transactional Analytics

► Hybrid OLTP and OLAP

Transactions in Massively Multiplayer Online Games

Kaiwen Zhang

Department of Software and IT Engineering,
École de technologie supérieure de Montréal,
Université du Québec, Montréal, QC, Canada

Synonyms

ACID properties in MMOGs; Consistency models in MMOGs

Definitions

Massively Multiplayer Online Games (MMOGs) are large virtual worlds replicated across player machines. Clients interact with the world by sending game actions, which must be executed over the world state according to specified game-aware ACID semantics.

Overview

Massively Multiplayer Online Games (MMOGs) are a popular genre of online games. These games revolve around large virtual worlds where players control their avatar to interact with others and the environment. The appeal of MMOGs lies in the persistence of data: the state of the game is constantly evolving and shared by the players. Personal data has value: most of the time spent in the game involves collecting items.

Another important aspect of MMOGs is their size. The millions of players which constitute the user base of commercial games are divided into shards, each with a limited capacity measured in thousands of clients. The focus is put on expanding the scale of games which must be supported by the underlying system.

The typical structure of MMOGs has clients each controlling a single character, or avatar, in the world through a graphical user interface. This game world is populated by various types of

objects: characters that are either controlled by players or artificial intelligence, noninteractive objects (e.g., trees, buildings) which form the environment, and mutable items which can be used or interacted with by characters. Players also take *actions*, which constitute the basic mean of interaction in MMOGs. This includes moving in the world, picking up items, dropping them from the character's inventory, and trading with other players. Since the game world is common to all players, the actions executed by one should be observable by others.

MMOGs can be treated as a database application (Gupta et al. 2009). The world objects are the data, and actions are logical sequences of read and write operations on the attributes of one or more objects. Ideally, these actions run as transactions with ACID properties. Actions require atomicity, executing in their entirety or not at all. Durability is essential as these game worlds run for long periods of time and need to survive various system failures. Different players can request actions on the same objects concurrently, requiring concurrency control to provide isolation.

However, several reasons exist as to why MMOGs cannot be implemented simply as a set of transactions using a traditional database system. First, most actions are update transactions, and the high rate of requests to be processed often cannot be handled by a single database server using traditional database techniques (Dalton 2007). Secondly, the system is, by nature, highly replicated. Each player sees part of the game world and the characters residing in that area. This is usually achieved by giving each client copies of the relevant game objects which are updated whenever changes occur (Kienzle et al. 2009).

Since these games can have thousands of players playing simultaneously, the update rate and the degree of replication are considerable. Propagating every change to every copy in an eager fashion (within transaction boundaries) in order to provide a consistent view to every player is simply not possible (Bharambe et al. 2006). In many cases, existing game engines restrict the number of players that can populate

a game region or instantiate the game to reduce the number of updates (Glinka et al. 2008). Furthermore, they commonly allow many inconsistencies to occur which the players are forced to accept (Henderson 2001). Durability is mostly handled in a very optimistic fashion: any gameplay shortly before an outage can be lost (Gupta et al. 2009).

Key Research Findings

Due to the unique replicated nature and workload properties of MMOGs, a set of consistency models are proposed which take into account game action semantics (Zhang and Kemme 2011). Consistency requirements can differ widely between actions due to their variation in complexity. An action's complexity is influenced by some of the following factors:

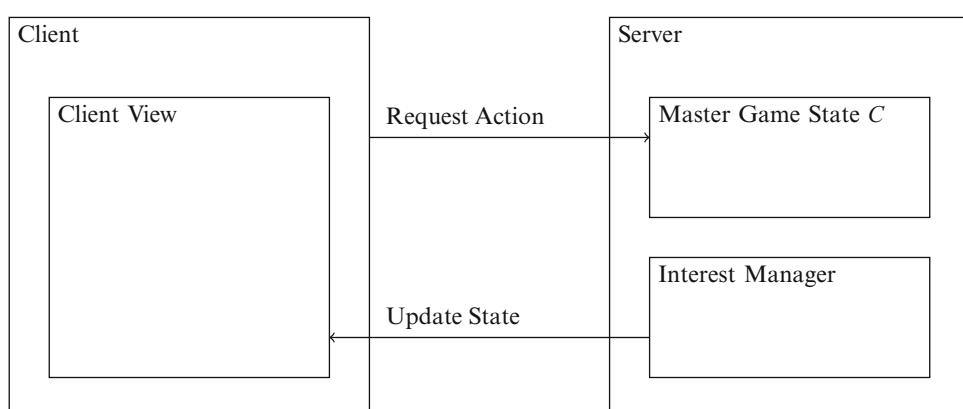
1. Repeatability: if an action is frequently executed, it might not require a high level of consistency as it can tolerate transient faults;
2. Number of objects: Atomicity becomes a greater issue when an action involves more than one object;
3. Number of reads: The effects of certain actions depend on the value of read attributes. Isolation is more difficult to maintain as the number of reads increases;

4. Importance: Crucial events with a significant impact on the game, actions that take a long time to be accomplished, or ones using real currency necessitate a high level of consistency.

Transactions in MMOGs have similarities with the various degrees of isolation, such as read committed and serializability, that are offered by traditional database systems (Berenson et al. 1995). In both cases, the developer is presented with a trade-off between performance and consistency. The approach in MMOGs goes further as it considers staleness of replicated objects.

Replication Model

The typical architecture of an MMOG system is depicted in Fig. 1. Each player hosts the client software of the game. It is able to render the game world and to receive input from the player. Clients control their avatar and can submit actions. The client software sends them to the server who serializes them, adjusts the game state, and notifies all players that are affected by the game change. This can be supported using a primary copy replication mechanism with master/replica proxy objects (Kienzle et al. 2009; Bharambe et al. 2006). Using this replication model, the game world consists of a set of objects, each of them being a collection of attributes. The server holds a *master proxy* (shortened to master) of



Transactions in Massively Multiplayer Online Games, Fig. 1 Simplified architecture

each object. The attribute values of the master proxy contain the latest and correct state of the corresponding object. The game state C , as maintained by the server, is defined as the union of the states of the master proxies of all objects.

Each client can then hold a read-only *replica proxy* (also called *replica*) for an object (white shapes in the Figure). At any given time, the state of a replica proxy might be lagging behind that of the corresponding master proxy, missing some of the changes that have already been installed at the latter. A client holds replica proxies only for objects that it is interested in. The composition of a client's *replication space* is determined dynamically by a process called *interest management*. Various methods exist for determining a player's interest (Boulanger et al. 2006). For example, one can define a circle around a player: all objects within this range are considered interesting and must be included in the replication space. The perceived state of all replicas held by a client is called the *client view*.

Action Definition and Processing

An action is a sequence of read and write operations performed on attributes of objects found in the world. The write set $WS(A)$ is defined as the set of all objects updated by action A . Without loss of generality, actions contain at least one write and do not create new objects.

As part of an action A , the client reads the state of some of its replicas before submitting the action to the server. The server then performs a sequence of read and write operations on some of the masters. As a result of the action, the master proxies of the objects in $WS(A)$ now possess a new state which must be disseminated to the replica proxies.

Figure 1 illustrates action execution. The client submits the action to the server, which executes it and disseminates the changes to all replicas, including the original client who submitted the action.

Read Operations

Understanding what objects are actually read and how the reads are performed is crucial to recognizing the required levels of consistency.

Action reads. Consider the following actions. Moving an avatar first reads the current position of the character. Drinking a bottle checks its content attribute: the action succeeds only if there is water remaining.

Action reads operations are inside the action code. Some are first performed on the client replicas. In particular, *validation reads* are used to verify whether an action is possible. For example, when a player tries to pick up an item, the local client software first checks the local replica state of the item. If it is on the ground, it forwards the action to the master. Otherwise, it rejects it immediately. The server repeats the validation reads on the master state, as it can differ from the client view. Client reads are considered preliminary, used to prevent sending unnecessary actions to the server. However, a valid action at the client might fail at the server. For instance, although a player saw an item on the ground, the pickup request might not succeed because the state at the server indicates that the item is already in the inventory of another avatar. In general, it is important to note that action reads performed on replicas usually always have the potential of being stale.

Client reads. A player continuously observes the state of all its replicas, whether they are stale or not. The player uses this information to decide what action to perform. The player is reading certain semantically relevant attributes before initiating an action on its client software. The key aspect is that any value from any object visible to the player can potentially be read and affect the judgment of the player. These reads are implicitly outside of the action code. Thus, each action has a *client read set*, determined by the player, which is read before the action is submitted. For instance, consider a player who decides to move to a location to pick up an item. Clearly, the move action does not read or update the item, only the position attribute of the player. Yet, the decision to move the player is directly dependent on the position of the item, as established by the player. The action read set contains the current position of the player's avatar, and the client read set contains the location of the item.

Transactions

In principle, a transaction for an action can be split into three parts. Clients first perform action reads and client reads on its replicas. The server then executes read and write operations on the master proxies. Finally, clients apply the changes from the server.

If possible, the entire transaction should be executed under full transactional properties: atomicity, isolation, and durability. Unfortunately, this would be costly, requiring distributed locking, eager propagation of updates, and a distributed commit protocol. No game architecture uses such approach. Instead, consistency is relaxed by considering correctness criteria for the server transaction and those for the client transactions separately.

Server Transactions

To begin, assume that actions are designed in a consistent manner: given a consistent game state C , the full execution of an action at the server under isolation transitions C to a new consistent state C' .

Moreover, the traditional definition of isolation is used for server transactions. A concurrency control mechanism is put in place to regulate parallel action execution. For instance, two-phase locking can be used to guarantee serializability (Lupei et al. 2010). Assume one action breaks a bottle and a concurrent action picks it up. If the break is serialized before the pickup, then a broken bottle is picked up. If the pickup gets the lock and executes before the break, then the break will not succeed as a bottle that is in the inventory of another avatar cannot be broken.

Formally, *master isolation* is provided if the execution at the server is serializable. That is, the concurrent execution of a set of actions at the server is equivalent to serial execution of these actions.

Master atomicity is said to be maintained if all or none of the action succeeds at the server. This can be implemented easily via a standard rollback functionality. Atomicity is more challenging in a distributed setting.

Durability is also a property maintained at the server and is dependent upon the aforementioned properties.

Providing transactional properties at the server guarantees that the game world as a whole remains consistent.

Client Transactions

The atomicity and isolation properties of client transactions reflect the player's gameplay experience.

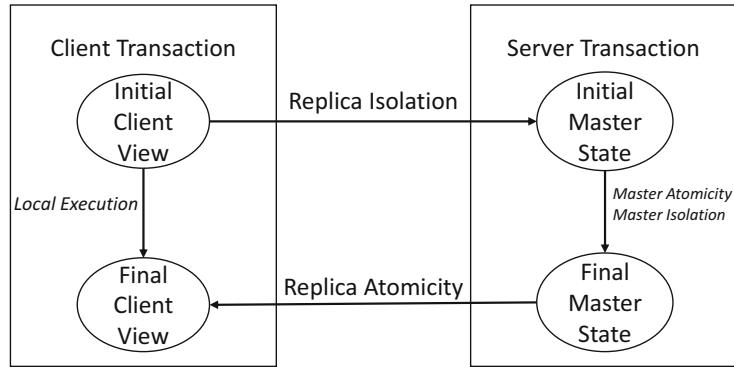
Replica atomicity is given if every update performed at a master is also applied at the related replicas. For performance reasons, this happens after the action commits at the server and not within the boundaries of the server transaction. Note that this does not require that the updates are applied within the boundaries of the server transaction. That is, the definition of atomicity is more relaxed than the traditional definition.

Replica isolation refers to the notion that if a client initiates an action based on its local action and client reads, then the updates triggered by the action from the server which are applied to the local replicas of the client are updates that conform to said reads. That is, if the client transaction had acted in isolation (performing only local reads and changing the local replicas), the outcome would be the same as the updates triggered by the server transaction.

For example, suppose a player wants to purchase an item at a given price. The client reads the price from the item replica as value P . If replica isolation is maintained, the price read at the master should also be value P . Suppose not and the master value is actually $P + 1$. Assuming the player has enough funds to purchase the item at the higher price, the update triggered from this action notifies the client that the item was bought at $P + 1$ and not P . Since the outcome is different from an execution using the locally read price value, replica isolation is violated.

Looking at this formally, replica isolation does not provide serializability at the global level. The state of the master might be different to the state read at a replica, leading to something similar to non-repeatable reads. However, if the difference does not have an effect on the write operations,

Transactions in Massively Multiplayer Online Games, Fig. 2
Client and server ACID properties



then it does not violate replica isolation. This means that the definition of replica isolation considers the game semantics.

Figure 2 illustrates the relationship between client and server transactions. When the server receives an action request, master atomicity and isolation enables the game state to move from one consistent state to the next. The view perceived at the client depends on whether replica isolation and atomicity are supported or not.

Examples of Application

Given the variety of actions, a multilevel approach to game consistency provides the best trade-off between consistency and performance. Lower categories are more efficient and scalable, while higher levels offer more consistency guarantees which are clearly defined. Thus, application developers can choose for each of their actions the proper consistency level, making them aware of the consistency properties that are achieved. In total, there exist five categories, each with an action handling protocol that can satisfy the requirements of each category while maximizing its performance and scalability.

All of the categories defined for a single server system offer master isolation and atomicity. Master isolation is achieved through a standard strict two-phase locking protocol. The three lowest category levels do not offer replica isolation and differ on replica atomicity. The three highest levels all offer replica atomicity but contrast in the way they handle replica isolation.

The granularity for read and write operations are object attributes and not entire objects to allow for fine-grained concurrency. Attributes are categorized by the way they are updated. An attribute x is said to be of category c if c is the *lowest* category level for which there exists an action which writes to x .

No Consistency

The lowest consistency category provides no guarantee and follows a “best-effort” *maybe* semantic model. Replica atomicity and isolation are not guaranteed. The server is not required to inform clients about any state changes incurred by no consistency actions.

This consistency category can be used for minor actions whose effects others do not necessarily need to be aware of. These actions are usually easily repeatable to compensate failures. For instance, graphical effects such as rotating a character or showing an emote (such as dancing or smiling) fall in this category as they do not have any real importance on gameplay.

Actions can be sent using an unreliable message channel (e.g., UDP).

Low Consistency

Like no consistency, low consistency does not require every update to be propagated to every replica, but it does provide a bound on the staleness of the data. Replica atomicity is therefore provided to some degree. This is useful for large volume actions that need to provide some guarantee in regard to their visibility.

The most common action type that fits into this category is the progressive movement of an avatar leading to a target destination. The idea here is that while clients might not perceive the exact position of a player at any time, they always know its approximate location, possibly defined through a specific error bound. For instance, the position seen at any replica should not be off by more than x movements or a distance of y units from the current position at the server. Another possibility is to support eventual convergence to the real position value.

Medium Consistency

The medium consistency category reflects the way current game middleware handles most of the actions. Replica atomicity is provided and requires exact updates to be sent. According to the definition, it is enough to send these updates after the action committed at the server. The bound on replica staleness is therefore the maximum latency for delivering an update from the server to a client.

Actions at this level still do not require replica isolation. Since clients can perform stale reads, the outcome at the server may be different to what the client expects.

This consistency level is appropriate for actions which are important enough to require exact updates. The inconsistencies which arise from the lack of replica isolation must be tolerable or predictable. For example, two nearby players send a request to pick up an item. At the server, one action is serialized before the other. The second action will be rejected because the item will already be picked up. Upon receiving this result, the client of the second action can justify the outcome due to the close proximity of the other player, as it can deduce that other player must have picked up the object.

High Consistency

Unexpected outcomes are sometimes undesirable. This is the case if critical attributes are affected by an action. For example, price is a critical attribute when buying an item. Normally, one does not want to buy an item for more than the locally visible price, but it is acceptable if

the item is cheaper. With medium consistency however, the item will be bought independently of the exact price at the server as long as the client has enough funds.

Thus, the high-consistency category introduces the concept of *critical attributes* and guarantees a limited form of replica isolation on such attributes. The isolation property is relaxed along two dimensions.

First, strict repeatable reads are not required; *comparable reads* are sufficient: the difference between the value at the master and the value at the replica must be *acceptable* for the action. What is acceptable depends on the game semantics. For instance, if a client submits a buy operation based on a certain price of an item, then a negative price difference (client price – server price) is not acceptable, and the action should fail as a repeatable read is violated. However, the action should succeed with a positive price difference.

Second, only replica isolation *safety* is provided. If the difference is unacceptable, the action must be rejected. High consistency essentially reduces any inadmissible executions into a rejection, which does not generate any inconsistencies since no changes are made to the game state.

Exact Consistency

For high consistency, stale critical attributes can lead to frequent aborts at the server. Reading stale values at the client and making decisions on behalf of this data can be considered an optimistic approach, where the final validation at the server may fail.

Exact consistency is the level where full replica isolation is provided. Guaranteed fresh data is possible only when client confirmation is integrated into the action itself. This category is designed specifically for those complex actions with multiple client input steps. An example is trading where items and money are exchanged between two avatars. Given the complexity and the importance of this type of action for gameplay, performance is not that crucial. In contrast, all players need to have the accurate state for decision-making and want to be assured

that the action succeeds properly at all involved parties.

This model enforces a stronger condition than replica atomicity. In all lower categories, changes are sent asynchronously to all replicas. In contrast, with exact consistency, all involved players receive the updates within the boundaries of the transaction, i.e., eagerly. Thus, atomic execution on all replicas of involved players is provided.

Other ACID Properties

For the sake of brevity, this chapter only covers consistency for MMOGs transactions. For more details on atomicity, which is nontrivial in a distributed (multi-server) setting, and durability, please refer to the following paper (Zhang and Kemme 2011).

Future Directions for Research

Future works include extended ACID model for MMOGs, adaptive transaction models, and optimized protocols for each consistency level.

References

- Berenson H, Bernstein PA, Gray J, Melton J, O’Neil EJ, O’Neil PE (1995) A critique of ANSI SQL isolation levels. In: SIGMOD conference
- Bharambe A, Pang J, Seshan S (2006) Colyseus: a distributed architecture for online multiplayer games. In: International conference on networked systems design & implementation (NSDI)
- Boulanger JS, Kienzle J, Verbrugge C (2006) Comparing interest management algorithms for massively multi-player games. In: ACM SIGCOMM NetGames workshop
- Dalton B (2007) Online gaming architecture: dealing with the real-time data crunch in mmhos. In: Game developer conference
- Glinka F, Ploss A, Gorlatch S, Müller-Iden J (2008) High-level development of multiserver online games. Int J Comput Games Technol 2008:1–16
- Gupta N, Demers A, Gehrke J, Unterbrunner P, White W (2009) Scalability for virtual worlds. In: International conference on data engineering (ICDE)
- Henderson T (2001) Latency and user behaviour on a multiplayer game server. In: COST264 workshop on networked group communication, pp 1–13
- Kienzle J, Verbrugge C, Kemme B, Denault A, Hawker M (2009) Mammoth: a massively multiplayer game research framework. In: International conference on foundations of digital games (FDG), pp 308–315
- Lupei D, Simion B, Pinto D, Misler M, Burcea M, Krick W, Amza C (2010) Towards scalable and transparent parallelization of multiplayer games using transactional memory support. In: SIGPLAN PPoPP
- Zhang K, Kemme B (2011) Transaction models for massively multiplayer online games. In: 30th IEEE symposium on reliable distributed systems (SRDS 2011), Madrid, 4–7 Oct 2011, pp 31–40

Trip Context Inference

► Using Big Spatial Data for Planning User Mobility

Truth Discovery

Laure Berti-Équille

Aix-Marseille University, CNRS, LIS, Marseille, France

Introduction

In the era of Big Data, *volume*, *velocity*, and *variety* are commonly used to characterize the salient features of Big Data. However, the importance of *veracity*, the fourth “V” of Big Data, is now well-recognized as a critical dimension that needs to be assessed by joint solutions coming from various research communities such as natural language processing (NLP), database (DB), and machine learning (ML), as well as from data science practitioners and journalists (Cohen et al. 2011; Berti-Équille 2016). The problem of estimating veracity of online information in presence of multiple conflicting data is very challenging: information extraction suffers from uncertainties and errors; information sources may be dependent or colluded; and misinformation is evolving and spreading fast in complex social networks. All these aspects have to be well-understood to be properly modeled in order to detect and combat effectively fake news and misinformation campaigns.

Rumor detection, misinformation spreading truth discovery, and fact checking have been the subjects of much attention recently (see recent surveys (Berti-Équille and Borge-Holthoefer 2015; Li et al. 2015; Berti-Équille 2016) and comparative studies (Waguilh and Berti-Equille 2014)). Typically, the main goal of truth finding is to infer the veracity of online (conflicting) information being claimed and/or (repeatedly) amplified by some sources on the Web, the blogosphere, and the social media.

More and more sophisticated truth discovery model algorithms have been designed to capture the richness and complexity of real-world fact-checking scenarios, both for generic and specific cases. Monitoring and tracking systems have been developed and tested in operational contexts e.g., “Fact Check” in Google News for fact checking (e.g., ClaimBuster Hassan et al. 2017), or tracking the social dynamics of online news sharing (e.g., Hoaxy (Shao et al. 2016)).

In this chapter, we will review these lines of work and also touch upon some cutting-edge problems for discovering truth in settings where information from multiple sources is conflicting and rapidly evolving. We discuss how close we are to meeting these challenges and identify many open problems for future research.

Definitions

Reputation, trust, and trustworthiness are concepts closely related to truth discovery.

Trustworthiness was originally measured by checking whether the contributor (i.e., the source) was contained in a list of trusted providers, since there exists an interdependency between the source (as data provider) and the data itself. On the one hand, data is likely to be accepted as true if it is provided by a trustworthy provider. On the other hand, the source is trustworthy if it provides true data. Thus, both data and data source can be checked to measure their trustworthiness (Dong et al. 2016). However, the assumption that the value confidence can be measured using only its source trustworthiness has some limitations: for example, non-authoritative sources may provide

some true and relevant information, whereas reputable sources may provide false or erroneous information. Therefore the users have to make decisions based on factors such as some prior knowledge about the subject, the reputation of the source.

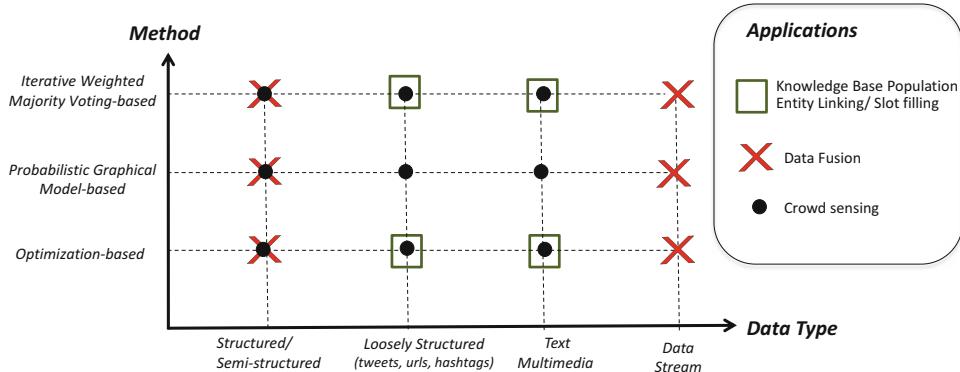
As we will see in the next section, the conjecture of SourceRank (Balakrishnan and Kambampati 2011) has influenced many research work in truth discovery: typically, in SourceRank, the more true relevant tuples a source returns, the more likely that other independent sources agree with its results. Inversely, independent sources are not very likely to agree on the same false tuples. Various methods have been then proposed to compute:

- Source trustworthiness as a score that quantifies how reliable the source is: It is a function of the confidence of its claims (also referred as source accuracy, quality, or reliability)
- value confidence as a score that quantifies the veracity of the claim. It is a function of the trustworthiness of the sources claiming it.

Current approaches for evaluating the veracity of data are iterative methods that compute and update each source trustworthiness score and then update the belief score of their claimed values.

Classification and Evolution of Truth Discovery Methods

We propose a classification of existing approaches across a two-dimensional space as illustrated in Fig. 1. Figure 1 presents a high-level view of the coverage of current methods in terms of applications such as data fusion, knowledge base population, and social sensing with crowd sourced data. The X-axis presents the considered data types in four categories based on the data structure: structured and semi-structured data, loosely structured and microtexts, free text or multimedia content, and data streams. The Y-axis defines the main principle of the underlying truth discovery methods such as weighted majority voting, probabilistic methods, or optimization-based methods as we will detail as follows.



Truth Discovery, Fig. 1 Classification of the main approaches for truth discovery with typical application contexts, underlying computation model and data type

- **Weighted voting-based methods** compute the value labels (i.e., true or false) and confidence scores using some variant of majority voting (MV) (Galland et al. 2010; Pasternack and Roth 2010; Yin and Han 2007). Traditional MV regards the value claimed by the majority of sources as the true value and randomly selects a value in case of tie with $1/|V_{o_a}|$ chance to infer the truth wrongly (with $|V_{o_a}|$, the number of distinct, conflicting values for the considered attribute a of object o). The reason is that MV assumes that each source has the same quality, but in reality, sources have different qualities, coverage, and scope. Various methods have adapted MV such that truths can be inferred through weighted voting to follow the principle that the information from reliable sources will be counted more in the aggregation.
- **Bayesian and graphical probabilistic model-based methods** were proposed to precisely address the issues of MV mainly related to the latent (unknown) properties of the sources and of the claims. TRUTHFINDER (Yin and Han 2007) was the first method that applies Bayesian analysis to estimate source trustworthiness and identify true claims with taking value similarity into consideration. DEPEN AND ACCU models (Dong et al. 2009, 2010) were also the first Bayesian models that incorporate source copying detection techniques, highlighting the importance of

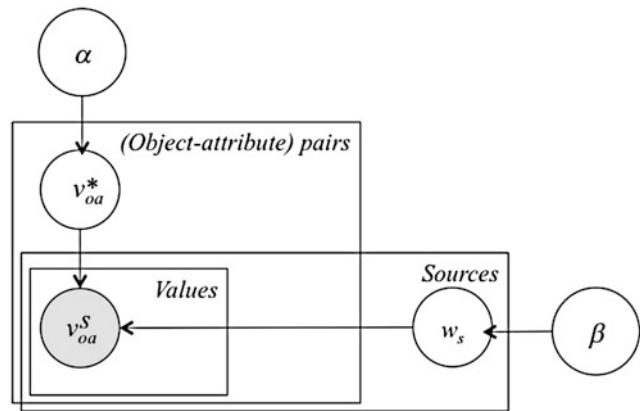
source dependence in truth discovery. Other methods (Zhao et al. 2012; Wang et al. 2012; Pasternack and Roth 2013) modeled a truth discovery scenario as a probabilistic graphical model (PGM) which expresses the conditional dependency structure (represented by edges) between random variables (represented by nodes). Figure 2 shows the generic PGM for truth discovery. In these approaches, the unknown identified truth noted $\{v_{o_a}^*\}$ and weights of the sources $\{w_s\}$ are the latent variables with a prior about the truth, denoted by α and a prior about the source reliability, denoted by β . The main principle is the following: if claimed values are close to the identified truth, the sources supporting them will gain weights. The goal is to maximize the likelihood and estimate high source weights when claims are close to the ground truth. To infer the two latent variables, techniques such as expectation maximization (EM) are adopted (Wang et al. 2012; Pasternack and Roth 2013), and the corresponding likelihood of a value being true is defined as:

$$\prod_{s \in S} p(w_s | \beta) \prod_{o \in O} \left(p(v_{o_a}^* | \alpha) \prod_{s \in S} p(v_{o_a}^s | v_{o_a}^*, w_s) \right). \quad (1)$$

- **Optimization-based methods** rely on setting an optimization function that can capture

Truth Discovery, Fig. 2

Generic plate diagram for GPM-based truth discovery methods



the relations between sources' qualities and claims' truth with an iterative method for computing these two sets of parameters jointly. The optimization function is generally formulated as:

$$\arg \min_{\{w_s\}, \{v_{oa}^*\}} \sum_{o \in O} \sum_{s \in S} w_s d(v_{oa}^s, v_{oa}^*) \quad (2)$$

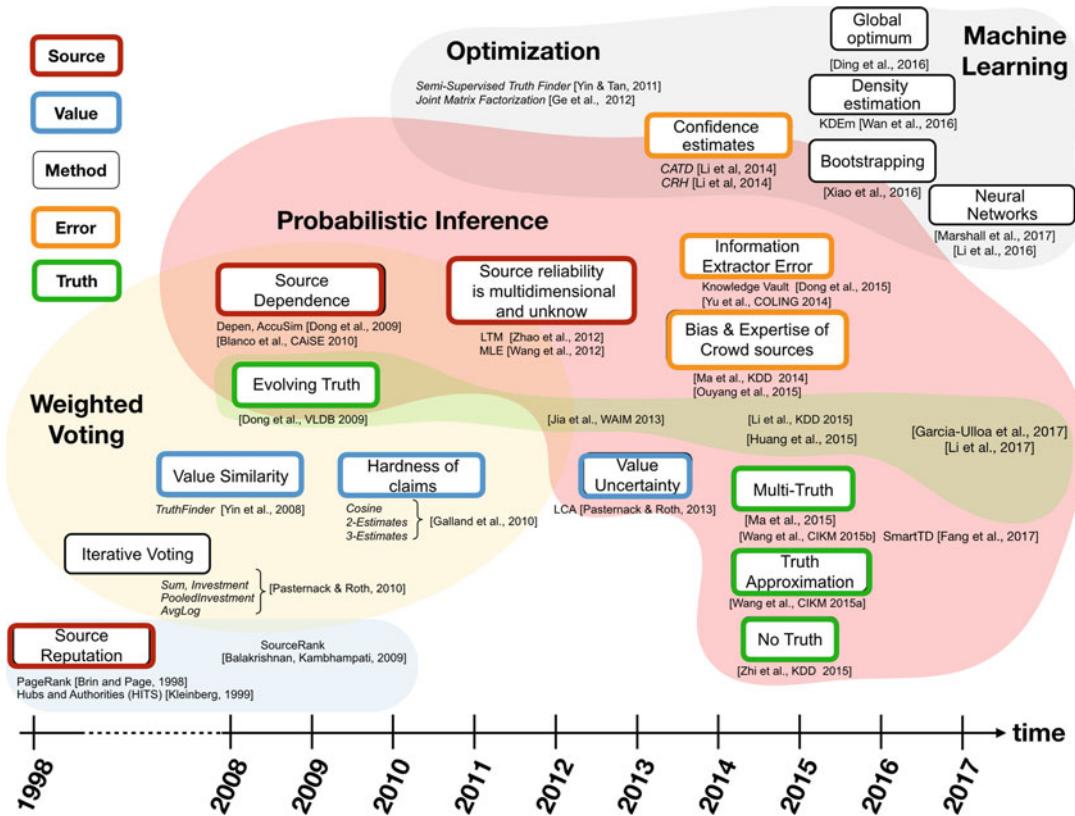
where $d(\cdot)$ is a distance function that measures the difference between the information provided by source s and the identified truth (e.g., 0–1 loss function for categorical data or L_2 -norm for continuous data). The objective function measures the weighted distance between the value for an object-attribute pair denoted $\{v_{oa}^s\}$ claimed by source s and the identified truth $\{v_{oa}^*\}$. By minimizing this function, the aggregated results will be closer to the information from the sources with high weights.

As illustrated in Fig. 3, the methods have significantly evolved in the last decade. The main trigger of this evolution is to overcome some of the limitations of previous approaches and to relax some modeling assumptions related to the sources (red boxes in the figure), their claimed values (blue boxes), the truth characteristics (green boxes), and the uncertainty or error associated (orange boxes) as we will mention in section “Modeling Considerations”.

Key Research Findings

We will now present the main principle and core algorithm of truth discovery and review the modeling assumptions underlying the key research findings in this domain. Truth discovery methods applied to structured data take as input some conflicting quadruplets in the form of $\{\text{source}, \text{object}, \text{attribute}, \text{value}\}$, where $\text{source} (s \in S)$ denotes the location where the data originates, $\text{object} (o \in O)$ is an entity, attribute is an attribute of the object, and $\text{value} (v \in V_{oa}^s \subset V)$ is the value of an attribute of an object claimed by a source. For example, a quadruplet: (imdb.com, director of Star Wars: Episode VIII - The Last Jedi, Rian Johnson) indicates that the website “IMDb” claims that the director of the movie “Star Wars: Episode VIII - The Last Jedi” is “Rian Johnson.” If a is a single-valued attribute for object o , $|V_{oa}^s| = 1$. In case of a multi-valued attribute, e.g., “the list of actors” or “full cast and crew”, $|V_{oa}^s|$ is bigger than 1.

For each input quadruplet, the methods infer a Boolean truth label (i.e., true or false) as the output. Formally, we name the factual value of an attribute a of an object o as the ground truth, denoted by v_{oa}^* . We note $(v)_m$ the label output given by a truth discovery method m for value v as the identified truth. Each method outputs the identified truth for each object and its set of attributes. We denote the identified truth labels of all objects' attributes in O output by method m as (V_{oam}) and the ground truth of v_{oa}^* . The



Truth Discovery, Fig. 3 Evolution of truth discovery research

closer (v_{o_a})_m is to $v_{o_a}^*$ for each attribute of each object, the better the method m performs. In most cases, the ground truth provided with real-world datasets is only a subset of the complete ground truth due to the prohibitive cost of collecting ground truth data.

Core Iterative Algorithm

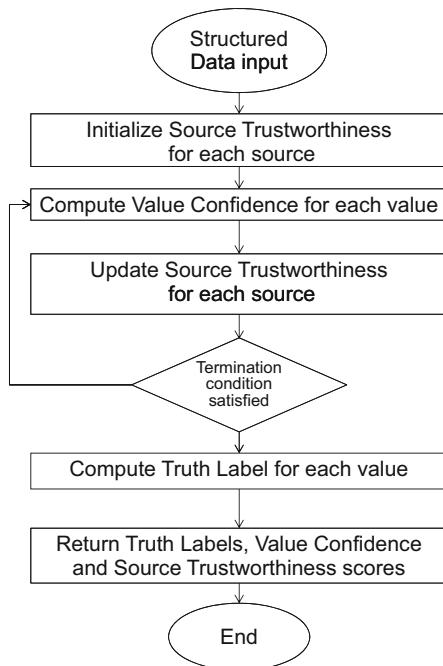
As illustrated in Fig. 4, the core algorithm of a fact finder is an iterative, transitive voting algorithm. First, the trustworthiness of each source is initialized. For each object and each attribute value, the method calculates the confidence from the reliability of its sources. Then, it updates the trustworthiness of each source from the confidence of the claims it makes. The procedure is repeated until the stopping condition is satisfied. Usually, the stopping criterion is defined either as a maximum number of iterations or a threshold under which the results (i.e., value confidence or

source trustworthiness scores) are considered to be stable from one iteration to the next. Some methods start with initializing the confidence scores of the values instead of the source trustworthiness, then compute source trustworthiness and update value confidence scores in a similar way. The methods differ in the way they compute and update the two scores as presented in section “Classification and Evolution of Truth Discovery Methods”.

Modeling Considerations

Every truth discovery method relies on several modeling assumptions related to the sources, the input claims, the truth, and the output result.

- **Modeling the sources.** Three main modeling assumptions concern the sources: (1) Sources are assumed to be self-consistent and nonredundant. This means that a source



Truth Discovery, Fig. 4 Basic iterative algorithm for truth discovery

should neither claim conflicting values for the same object-attribute pair nor provide duplicate claims; (2) current methods rely on trusting the majority, and they are adapted only when the sources are assumed to be predominantly honest and the number of sources providing true claims is assumed to be significantly larger than the number of sources providing false claims. This assumption referred in Waguilh and Bertie-Equille (2014) as the “optimistic scenario” is an important limitation for nowadays truth discovery scenarios.

- **Modeling the input claims.** Several considerations are important: (1) The cardinality (e.g., single or multi-valued attributes) and type of input claims (e.g., nominal vs numeric data) will determine the methods that can be applied and if data formatting is required; (2) the hardness of evaluating certain claims veracity can be considered in the model (see Galland et al. 2010); (3) the extraction of structured claims is generally assumed to have no error, but

information extractors can generate uncertainties to be considered in truth discovery computation (see Dong et al. 2014); (4) only claims with a direct source attribution are considered in the current methods. This requires that each claim has its source explicitly identified with a direct correspondence; (5) claims are usually assumed to be positive (except in Galland et al. 2010): e.g., *S claims that A is false* or *S does not claim A is true* are not considered by current methods; (6) regarding claim uncertainty, few methods (e.g., LCA Pasternack and Roth 2013) can handle cases such as *S claims that A is true with 15% uncertainty*. They use a weight matrix to express the confidence of each source in its assertions.

- **Modeling the relationship between source reliability and claim truthfulness.** (1) A potential limitation of current approaches is that the relationships between source reliability and claim truthfulness are often represented by simplified functions (e.g., linear, quadratic, or binomial). This assumption may lead to suboptimal truth discovery results because the exact relational dependency between sources and claims is usually unknown *a priori*; (2) another important related assumption is that the probability a source asserts a claim is independent of the truth of the claim. For example, some sources may choose to stay silent except for trivial or “easy” truths. This will lead to a very high trustworthiness score for these sources, although they may not deserve it. Penalties in these cases can be applied to refine some existing model in order to address the “long-tail phenomenon” (Li et al. 2014).
- **Modeling choices for the truth and output results.** (1) Each claim is assumed to be either true or false. Although value confidence score is generally returned as output by every method, truth labeling is usually Boolean with no contextual information or additional evidence (except in Waguilh et al. 2015 where a posteriori explanations of the truth discovery results are provided in the form of decision trees and the robustness of a true claim is tested with the generation of allegations); (2) single-truth assumption: Claims related to the

attribute of an object are organized into disjoint mutual exclusion sets where only one of the claims in each set is generally assumed to be true. However, some methods can handle multi-truth scenarios (e.g., LTM Zhao et al. 2012) and situations where none of the values claimed by the sources is actually true (Zhi et al. 2015).

- **Evaluation limits.** To evaluate efficiency and effectiveness performance of truth discovery methods, various metrics are traditionally used. Memory cost and running time can be used to evaluate the efficiency. Recall, precision, F1-measure, accuracy (or error rate) for categorical data, mean of absolute error (MAE), and root of mean square error (RMSE) for continuous data are computed for quality performance when ground truth is available. However, the labor cost of collecting ground truth is usually prohibitive. As a consequence, ground truth is often very limited or even impossible to obtain in a reasonable amount. Methods that show the same accuracy on sparse ground truth may have different performance in reality. Under these circumstances, it is hard to conclude which method performs better as we cannot trust the comparison results due to its low statistical significance over sparse ground truth (Berti-Équille and Borge-Holthoefer 2015; Waguilh and Berti-Equille 2014).

Relaxing these modeling assumptions, enlarging the range of real-world scenarios captured by the models, evaluating and benchmarking truth discovery methods, and estimating the significance of the evaluation results based on sparse ground truth are the next challenging milestones in the truth discovery research agenda.

Novel Research Directions

Novel approaches leveraging machine learning have been recently proposed. They are promising research directions for addressing the great diversity of real-world misinformation scenarios.

ETCIBoot (Estimating truth and confidence interval via bootstrapping) (Xiao et al. 2016). Existing truth discovery methods focus on providing a point estimator for each object’s truth, but in many real-world applications, confidence interval estimation of truths is more desirable because it contains richer information. ETCIBoot constructs confidence interval estimates as well as identifies truths with integrating bootstrapping techniques into the truth discovery procedure. Due to the properties of bootstrapping, the estimators obtained by ETCIBoot are more accurate and robust compared with the state-of-the-art truth discovery approaches. The authors of Xiao et al. (2016) theoretically prove the asymptotic consistency of the confidence interval obtained by ETCIBoot.

Neural networks. A novel neural network-based approach has been recently proposed by Marshall et al. (2018). This method can learn complex relational dependency between source reliability and claim truthfulness. A multi-layer neural network model has been developed to solve the truth discovery problem in social sensing without any assumption on the prior knowledge of the source-claim relational dependency distribution. In particular, a neural network for truth discovery is defined by a set of input neurons which are activated by social sensing data (i.e., sources and the claims they make). After being weighted and transformed by a learning function, the activation of these neurons are then passed on to other neurons inside the neural networks. This process is repeated until the output neuron that determines the truthfulness of a claim is activated. The complex source-claim relational dependency is learned by the neural network model through the above training process.

Conclusions

This chapter presented an overview of recent advances of truth discovery research emphasizing on the main methods and their underlying modeling assumptions.

We have observed that none of the methods constantly outperforms the others in terms of precision/recall, and a “one-fits-all” approach does not seem to be achievable. Most of the current methods have been designed for excelling in *optimistic* scenarios with a reasonable number of honest and reliable sources. However, experiments (e.g.) in Waguih and Berti-Équille (2014) revealed that, for *pessimistic* or adversarial scenarios when most of sources are not reliable, most of the methods have relatively low precision and some expose prohibitive runtime and may suffer from scalability or fluctuating results. Ensembling (Berti-Équille 2015) and bootstrapping (Xiao et al. 2016) truth discovery methods seems to be very promising research directions.

Another challenge is related to the usability of the methods. The assumptions made by current truth discovery models and their complex parameter settings make most methods still difficult to apply to the wide diversity of on-line information and existing scenarios on the Web. Since limited and generally sparse ground truth is available, performance evaluation and comparative studies may not be reliable and have low statistical significance; to this matter, benchmarks and repeatability experiments are critically needed.

Cross-References

- ▶ Data Fusion
- ▶ Data Integration

References

- Balakrishnan R, Kambhampati S (2011) SourceRank: relevance and trust assessment for deep web sources based on inter-source agreement. In: Proceedings of the international conference on world wide web (WWW 2011), pp 227–236
- Berti-Équille L (2015) Data veracity estimation with ensembling truth discovery methods. In: 2015 IEEE international conference on big data, big data 2015, Santa Clara, 29 Oct–1 Nov 2015, pp 2628–2636
- Berti-Équille L (2016) Scaling up truth discovery. In: Proceedings of the 32nd IEEE international conference on data engineering (ICDE), Helsinki, 16–20 May 2016, pp 1418–1419
- Berti-Équille L, Borge-Holthoefer J (2015) Veracity of data: from truth discovery computation algorithms to models of misinformation dynamics. Synthesis lectures on data management. Morgan & Claypool Publishers, San Rafael
- Cohen S, Li C, Yang J, Yu C (2011) Computational journalism: a call to arms to database researchers. In: Proceedings of the fifth biennial conference on innovative data systems research (CIDR 2011), pp 148–151
- Dong XL, Berti-Équille L, Srivastava D (2009) Integrating conflicting data: the role of source dependence. PVLDB 2(1):550–561
- Dong XL, Berti-Équille L, Hu Y, Srivastava D (2010) Global detection of complex copying relationships between sources. Proc VLDB Endow 3(1–2):1358–1369
- Dong XL, Gabrilovich E, Heitz G, Horn W, Lao N, Murphy K, Strohmann T, Sun S, Zhang W (2014) Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: The 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD’14), New York, 24–27 Aug 2014, pp 601–610
- Dong XL, Gabrilovich E, Murphy K, Dang V, Horn W, Lugaresi C, Sun S, Zhang W (2016) Knowledge-based trust: estimating the trustworthiness of web sources. IEEE Data Eng Bull 39(2):106–117
- Galland A, Abiteboul S, Marian A, Senellart P (2010) Corroborating information from disagreeing views. In: WSDM, pp 131–140
- Hassan N, Zhang G, Arslan F, Caraballo J, Jimenez D, Gawsane S, Hasan S, Joseph M, Kulkarni A, Nayak AK, Sable V, Li C, Tremayne M (2017) Claimbuster: the first-ever end-to-end fact-checking system. PVLDB 10(12):1945–1948
- Li Q, Li Y, Gao J, Su L, Zhao B, Demirbas M, Fan W, Han J (2014) A confidence-aware approach for truth discovery on long-tail data. Proc VLDB Endow 8(4):425–436
- Li Y, Gao J, Meng C, Li Q, Su L, Zhao B, Fan W, Han J (2015) A survey on truth discovery. SIGKDD Explor 17(2):1–16
- Marshall J, Argueta A, Wang D (2018) A neural network approach for truth discovery in social sensing. In: 2017 IEEE 14th international conference on mobile Ad Hoc and sensor systems (MASS), pp 343–347
- Pasternack J, Roth D (2010) Knowing what to believe (when you already know something). In: Proceedings of the conference on computational linguistics (COLING’10), pp 877–885
- Pasternack J, Roth D (2013) Latent credibility analysis. In: Proceedings of the international world wide web conference (WWW 2013), pp 1009–1020
- Shao C, Ciampaglia GL, Flammini A, Menczer F Hoaxy: a platform for tracking online misinformation. In: Proceedings of the 25th international conference companion on world wide web (WWW’16 companion), Republic and canton of Geneva, 2016. International World Wide Web Conferences Steering Committee, pp 745–750

- Waguih DA, Berti-Equille L (2014) Truth discovery algorithms: an experimental evaluation. CoRR abs/1409.6428
- Waguih DA, Goel N, Hammady HM, Berti-Equille L (2015) AllegatorTrack: combining and reporting results of truth discovery from multi-source data. In: Proceedings of the IEEE international conference on data engineering (ICDE 2015), pp 1440–1443
- Wang D, Kaplan LM, Le HK, Abdelzaher TF (2012) On truth discovery in social sensing: a maximum likelihood estimation approach. In: IPSN, pp 233–244
- Xiao H, Gao J, Li Q, Ma F, Su L, Feng Y, Zhang A (2016) Towards confidence in the truth: a bootstrapping based truth discovery approach. In: Proceedings of the 22Nd ACM SIGKDD international conference on knowledge discovery and data mining (KDD’16), pp 1935–1944
- Yin X, Han J (2007) Truth discovery with multiple conflicting information providers on the web. In: Proceeding of 2007 ACM SIGKDD international conference on knowledge discovery in databases (KDD’07)
- Zhao B, Rubinstein BIP, Gemmell J, Han J (2012) A Bayesian approach to discovering truth from conflicting sources for data integration. PVLDB 5(6):550–561
- Zhi S, Zhao B, Tong W, Gao J, Yu D, Ji H, Han J (2015) Modeling truth existence in truth discovery. In: Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining (KDD’15), pp 1543–1552

Types of Stream Processing Algorithms

Lukasz Golab
University of Waterloo, Waterloo, ON, Canada

Synonyms

[One-pass algorithms](#); [Online Algorithms](#)

Definitions

A stream processing algorithm operates over a continuous and potentially unbounded stream of data, arriving at a possibly very high speed, one item or one batch of items at a time, and does so in limited time per item and using limited working storage. At any point in time, a stream algorithm can produce an answer over the prefix of the stream observed so far or over a sliding

window of recent data. Stream processing algorithms are used to answer *continuous* queries, also known as *standing* queries.

Stream processing algorithms can be categorized according to (1) what output they compute (e.g., what function is being computed, is the answer exact or approximate) and (2) how they compute the output (e.g., sampling vs. hashing, single-threaded vs. distributed, one-pass vs. several passes).

Overview

Stream processing algorithms operate sequentially over unbounded input streams and produce output streams. The input stream is assumed to consist of *items* which may be key-value pairs or structured tuples. Stream algorithms face the following common challenges.

- In contrast to offline algorithms, the input is not provided in advance, and instead it arrives continuously and sequentially, possibly at a very high speed, one item or one batch of items at a time. The data arrival order cannot be controlled by the algorithm as it is determined by the source (and perhaps also by the communication channel which may reorder the data).
- In order to keep up with the input, the algorithm must process items in limited time and using limited (and fast) working storage. Latency must be low as late results rapidly lose value. For example, in network and infrastructure monitoring, anomalies and attacks must be detected as quickly as possible.

Although they share the above characteristics, there is a wide variety of stream processing algorithms. They can be classified based on semantics (what is computed) and methodologies (how it is computed).

Semantic Classification

From the point of view of semantics, one can classify stream processing algorithms into the following types:

- What function or query is computed? Examples include identifying frequently occurring items and counting the number of distinct items.
 - Is the output exact or approximate? Many stream processing algorithms produce approximate answers as a consequence of using limited time and memory. However, some simple functions may be computed exactly. For example, to output the maximum value seen so far in a stream of numbers, it suffices to keep track of the current maximum value and compare it to every incoming value.
 - Is the output computed over the entire stream observed so far or over a sliding window?
- any parallel algorithm, load balancing is critical (Nasir et al. 2015). Additionally, some algorithms may be distributed by moving some computation to sources of the data (e.g., sensors or network interface cards).
- How many passes over the stream are allowed? In many applications, only one pass is allowed, so an algorithm only gets one look at the data. However, multi-pass algorithms have also been studied. These are useful in applications where a data stream has already been collected and can be played back several times or when a static dataset is processed sequentially offline.

Technical Classification

From a technical point of view, stream processing algorithm may be classified as follows:

- How is an unbounded stream “compressed” to fit in limited working storage? In high-volume stream processing, even sliding windows may be too large to store and process. Furthermore, the space of possible distinct items (*keys*) may be too large to fit in memory (e.g., all possible IP addresses). Stream processing algorithms may use *sampling* to decide which items to retain in working storage and *hashing* or *sketching* (Cormode 2017) to map the key space to a smaller domain. Of course, the use of sampling, hashing, or other lossy summarization techniques means that results will not be exact.
- Parallelism and distribution. Classical stream algorithms assume a single processing thread, but modern distributed stream processing infrastructures support inter- and intra-operator parallelism. In inter-operator parallelism, different operators are executed by different nodes in a computing cluster. In intra-operator parallelism, multiple instances of the same operator are running and are responsible for different parts of the input. For example, frequency counts may be computed in parallel by dividing the key space among threads. As is the case in

Key Research Findings

The three Vs of big data are volume, velocity, and variety. “Big streams” are characterized by high volume, fast arrival rates, and a wide variety of distinct items or keys (e.g., the number of distinct IP addresses in an Internet traffic stream). What follows is a summary of key research findings on algorithms for large-scale stream processing.

Stream Processing Algorithms

This section discusses fundamental algorithms for popular stream processing tasks: sampling from a stream, counting the number of distinct items (also known as *cardinality estimation*), and identifying frequently occurring items. Let d be the number of distinct items or keys in the stream and let n be the total number of items in the stream (of course n is not known in advance).

One of the earliest stream processing algorithms produced a random sample of r items from the stream (Vitter 1985). Random sampling is an important operation as it can be used to estimate various statistical properties of the stream. Note that if n is known, the sampling from the stream is easy: each item should be sampled with probability $\frac{r}{n}$. However, in an online setting where the stream arrives one item at a time, n is not known a priori.

The algorithm, called *reservoir sampling*, works as follows. The first r items that arrive are included in the sample. When the $r + 1$ st

item arrives, with probability $\frac{r}{r+1}$, the new item replaces a randomly selected item from the current sample (and with probability $1 - \frac{r}{r+1}$, the new item is ignored). More details about stream sampling may be found in Haas (2016).

Note that some data stream engines employ *load shedding* (Babcock et al. 2004; Tatbul et al. 2003) by randomly dropping some stream items during periods of overload and processing all items during steady state. However, this type of random load shedding does not produce a random sample of the stream.

Another early work on stream processing considered (approximately) counting the number of distinct items in the stream (Flajolet and Martin 1983) and then is improved in Durand and Flajolet (2003) and Flajolet et al. (2007). It is assumed that there are more distinct items than can fit in memory, and therefore it is not possible to compute the distinct count exactly by indexing the items.

The Flajolet-Martin algorithm requires only $\log(d)$ space. The main idea is to use a random hash function that maps stream items to bit vectors of length $\log(d)$. On average, one would expect half the items to hash to bit vectors that end with a zero and the other half to hash to bit vectors that end with a one. Similarly, on average, a quarter of the items should hash to bit vectors that end with two zeros and so on. In general, $\frac{1}{2^k}$ items should have to bit vectors that end with k zeros. To exploit this observation, the algorithm hashes each item and records the number of zeros the hash ends with. Let ℓ be largest number of such trailing zeros seen so far. It turns out that a reliable estimate for the number of distinct items is $\frac{2^\ell}{0.77351}$. To improve accuracy, multiple hash functions can be used, and the resulting estimates can be averaged.

Streaming algorithms for identifying frequently occurring items must also resort to approximation – in large-scale data streams, there may not be enough fast memory or enough per-item processing time to maintain frequency counters for all items. The challenge is to decide which item frequencies to track at a given point in time. Note that initially infrequent items may become frequent over the lifetime of stream. A

survey and an experimental comparison of well-known frequent item algorithms may be found in Cormode and Hadjieleftheriou (2010).

Two approaches for identifying frequent items have appeared in the literature:

- Counter-based approaches: allocate some number of counters to maintain the frequencies of selected items.
- Sketch-based approaches: maintain a small summary of the stream and use it to identify frequent items.

One of the earliest frequent item algorithms used a counter-based approach (Misra and Gries 1982). It uses k counters, each labeled with the item it represents, and identifies items that appear at least $\frac{n}{k}$ times. Initially, all counters are set to zero and their labels are blank. When a new item arrives with value v and there already exists a counter for v , that counter is incremented. Otherwise, if there is at least one counter with a count of zero, this counter now becomes allocated to v and is incremented. Otherwise, if all k counters are non-zero, then *all* k counters are decremented. A similar algorithm called *Space Saving* was proposed subsequently. It differs in how it handles the case when all k counters are non-zero: *Space Saving* selects the counter with the smallest count, gives it to v , and increments it (Metwally et al. 2005). Other examples of counter-based algorithms for finding frequent items (and quantiles) include Greenwald and Khanna (2001) and Manku and Motwani (2002).

On the other hand, the Count-Min sketch (Cormode and Muthukrishnan 2005) and earlier related sketches (such as Charikar et al. 2002) are examples of a stream summaries that can approximate the frequency distributions of items and therefore identify frequent items. These techniques use a set of independent hash functions to map newly arrived items onto a smaller space.

Sliding Window Algorithms

Let w be the sliding window size, either in the number of items or time units. Let s be the slide

interval of a window; e.g., a *time-based* window may have a size of 10 minutes, and it may slide forward every minute, whereas a *count-based* window may store the most recent 1000 items and may slide every ten items.

The stream processing algorithms described earlier had to deal with inserting new items. The additional technical challenge in sliding window processing is removing items that *expire* from the window. Notice that every item eventually expires, which happens w time units after its insertion or after w new items have arrived.

The remainder of this section discusses algorithms for sampling from sliding windows, computing approximate statistics using limited space, computing simple aggregates exactly using limited space, and joining multiple streams using sliding windows.

Sliding window sampling algorithms were presented in Babcock et al. (2002). For count-based windows, the idea is to reuse reservoir sampling. However, whenever an item is included in the sample, the index, or position within the stream, of its *successor* is preselected by randomly choosing an index from the next window (the w items that will arrive after the original item expires). For time-based windows, the idea is to assign a priority to each element, corresponding to a random number between zero and one. The sample then includes k non-expired items with the highest priority. Importantly, it is not sufficient to store just those k items. Additionally, the algorithm stores items for which there is no item with both a later timestamp and a higher priority. This ensures that any sampled item that expires can be replaced with another stored item. The sampling algorithms were later improved by Braverman et al. (2012).

There exist many algorithms for approximately computing various statistics over count-based sliding windows using sublinear space, including counts, sums, quantiles, and frequent items (Arasu and Manku 2004; Datar et al. 2002; Lee and Ting 2006). The idea is to partition a window into blocks or sub-windows and precompute some statistic for each block. When the window slides forward, new blocks are opened and old blocks are expired.

Approximation is due to the fact that the oldest block may contain some items that have already expired.

A special case has also been studied of computing simple group-by aggregates such as sum and average over count- or time-based windows. This can be done without storing all the items in the window. Again, the idea is to partition a window into buckets whose sizes are equal to the slide interval and pre-aggregate the items within each bucket. The final result is computed by merging the partial aggregates from each bucket, and old buckets are replaced with new buckets as the window slides. For example, to compute the sum of the values for each key, it suffices to add up the partial sums pre-aggregated in each bucket. Examples of this method, and related methods that maintain a hierarchical structure of buckets, can be found in, e.g., (Arasu and Widom 2004; Bulut and Singh 2005; Krishnamurthy et al. 2006; Li et al. 2005).

Since data streams are unbounded, joining two (or more) streams is usually done using sliding windows. When a new item arrives on one stream, it can only join with a window of items from the other stream. A simple pipelined hash join implementation maintains hash tables for both windows, and when a new item arrives on one stream, it is inserted into its window and the other window is probed for matching items. Hash tables can be partitioned into buckets to make data expiration easier. Examples of sliding window join algorithms may be found in, e.g., (Golab and Özsu 2003; Kang et al. 2003). A parallel implementation was proposed in Teubner and Müller (2011).

T

Distributed Algorithms

In data stream processing, distributed computation can refer to one of the following two scenarios:

1. Data streams continuously arrive, potentially from multiple distributed sources, to a stream processing system, which consists of multiple compute nodes. Each node may be responsible for processing different parts of the stream

- (e.g., different subsets of keys) or different query operators.
2. Data streams are continuously generated and partially processed by distributed sources. Pre-processed streams arrive to a stream processing system for final processing.

In the first scenario, distributed systems such as Spark Streaming have recently been proposed (Zaharia et al. 2013). These systems execute MapReduce-style jobs over (small) batches of data. Furthermore, many of the counter-based and sketch-based algorithms discussed earlier have a useful property that the underlying data structures are *mergeable* while preserving their error guarantees (Agarwal et al. 2013). That is, different processing nodes can work on different subsets of the stream and produce local summaries, and the final answer can be computed from the merged summary.

In the second scenario, some computation (and data volume reduction) can be done by the sources (sensors, network interface cards, etc.). This approach is useful when it is infeasible to send all the data for processing due to insufficient processing resource and/or insufficient bandwidth. There are two ways to realize this approach: pull-based and push-based. In pull-based methods, the system periodically requests new data from the sources to recompute the result or a function of interest. In push-based methods, the sources decide when to push updates to the system. The challenge is to determine a trade-off between push frequency (and therefore communication bandwidth) and data freshness. Push-based methods have been studied for computing statistics such as sums over distributed streams (Olston et al. 2003) and maintaining the k largest values in distributed streams (Babcock and Olston 2003). Furthermore, sampling over distributed streams has been studied by Cormode et al. (2012).

Examples of Applications

Internet traffic monitoring was an early motivating application: identifying source or destination

IP addresses that consume significant bandwidth, detecting attacks and malfunctions, etc. (Cranor et al. 2003). Sensor networks (Madden and Franklin 2002) and electronic financial trading (Stonebraker et al. 2005) were also mentioned. Recently, large-scale distributed stream processing systems have been built to analyze social media streams (e.g., Twitter's Heron Kulkarni et al. 2015) and Web search logs (e.g., Google's MillWheel Akidau et al. 2013), and there is recent interest in stream processing for energy analytics (Liu et al. 2017).

Future Directions for Research

A majority of existing work on stream processing algorithms focuses on computing functions and aggregates over numeric streams. Recently, there has been work on complex tasks such as online machine learning models. Another area for future research involves streaming graph processing, where the input consists of edges and nodes of a time-evolving graph.

Cross-References

- [Continuous Queries](#)
- [Definition of Data Streams](#)
- [Introduction to Stream Processing Algorithms](#)
- [Stream Processing Languages and Abstractions](#)
- [Stream Query Optimization](#)

References

- Agarwal PK, Cormode G, Huang Z, Phillips JM, Wei Z, Yi K (2013) Mergeable summaries. ACM Trans Database Syst 38(4):26:1–26:28
- Akidau T, Balikov A, Bekiroglu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) Millwheel: Fault-tolerant stream processing at internet scale. PVLDB 6(11):1033–1044
- Arasu A, Manku GS (2004) Approximate counts and quantiles over sliding windows. In: Proceedings of the twenty-third ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, 14–16 June 2004, Paris, pp 286–296

- Arasu A, Widom J (2004) Resource sharing in continuous sliding-window aggregates. In: (e)Proceedings of the thirtieth international conference on very large data bases, Toronto, 31 Aug–3 Sept 2004, pp 336–347
- Babcock B, Olston C (2003) Distributed top-k monitoring. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, 9–12 June 2003, pp 28–39
- Babcock B, Datar M, Motwani R (2002) Sampling from a moving window over streaming data. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, 6–8 Jan 2002, San Francisco, pp 633–634
- Babcock B, Datar M, Motwani R (2004) Load shedding for aggregation queries over data streams. In: Proceedings of the 20th international conference on data engineering, ICDE 2004, 30 Mar–2 Apr 2004, Boston, pp 350–361
- Braverman V, Ostrovsky R, Zaniolo C (2012) Optimal sampling from sliding windows. *J Comput Syst Sci* 78(1):260–272
- Bulut A, Singh AK (2005) A unified framework for monitoring data streams in real time. In: Proceedings of the 21st international conference on data engineering, ICDE 2005, 5–8 Apr 2005, Tokyo, pp 44–55
- Charikar M, Chen KC, Farach-Colton M (2002) Finding frequent items in data streams. In: Proceedings of 29th international colloquium automata, languages and programming, ICALP 2002, Malaga, 8–13 July 2002, pp 693–703
- Cormode G (2017) Data sketching. *Commun ACM* 60(9):48–55
- Cormode G, Hadjieleftheriou M (2010) Methods for finding frequent items in data streams. *VLDB J* 19(1):3–20
- Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. *J Algorithm* 55(1):58–75
- Cormode G, Muthukrishnan S, Yi K, Zhang Q (2012) Continuous sampling from distributed streams. *J ACM* 59(2):10:1–10:25
- Cranor CD, Johnson T, Spatscheck O, Shkapenyuk V (2003) Gigascope: a stream database for network applications. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, 9–12 June 2003, pp 647–651
- Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. *SIAM J Comput* 31(6):1794–1813
- Durand M, Flajolet P (2003) Loglog counting of large cardinalities (extended abstract). In: Proceedings of 11th annual European symposium algorithms – ESA 2003, Budapest, 16–19 Sept 2003, pp 605–617
- Flajolet P, Martin GN (1983) Probabilistic counting. In: 24th annual symposium on foundations of computer science, Tucson, 7–9 Nov 1983, pp 76–82
- Flajolet P, Fusy E, Gandouet O, Meunier F (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Proceedings of the conference on analysis of algorithms, pp 127–146
- Golab L, Özsü MT (2003) Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of 29th international conference on very large data bases VLDB 2003, 9–12 Sept 2003, Berlin, pp 500–511
- Greenwald M, Khanna S (2001) Space-efficient online computation of quantile summaries. In: Proceedings of the 2001 ACM SIGMOD international conference on management of data, Santa Barbara, 21–24 May 2001, pp 58–66
- Haas PJ (2016) Data-stream sampling: basic techniques and results. In: Garofalakis M, Gehrke J, Rastogi R (eds) Data stream management – processing high-speed data streams. Springer, Heidelberg, pp 13–44
- Kang J, Naughton JF, Viglas S (2003) Evaluating window joins over unbounded streams. In: Proceedings of the 19th international conference on data engineering, 5–8 Mar 2003, Bangalore, pp 341–352
- Krishnamurthy S, Wu C, Franklin MJ (2006) On-the-fly sharing for streamed aggregation. In: Proceedings of the ACM SIGMOD international conference on management of data, Chicago, 27–29 June 2006, pp 623–634
- Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel JM, Ramasamy K, Taneja S (2015) Twitter heron: stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, Melbourne, 31 May–4 June 2015, pp 239–250
- Lee L, Ting HF (2006) A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proceedings of the twenty-fifth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, 26–28 June 2006, Chicago, pp 290–297
- Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec* 34(1):39–44
- Liu X, Golab L, Golab WM, Ilyas IF, Jin S (2017) Smart meter data analytics: Systems, algorithms, and benchmarking. *ACM Trans Database Syst* 42(1):2: 1–2:39
- Madden S, Franklin MJ (2002) Fjording the stream: an architecture for queries over streaming sensor data. In: Proceedings of the 18th international conference on data engineering, San Jose, 26 Feb–1 Mar 2002, pp 555–566
- Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: VLDB 2002, Proceedings of 28th international conference on very large data bases, 20–23 Aug 2002, Hong Kong, pp 346–357
- Metwally A, Agrawal D, El Abbadi A (2005) Efficient computation of frequent and top-k elements in data streams. In: Proceedings of 10th international conference on database theory – ICDT 2005, Edinburgh, 5–7 Jan 2005, pp 398–412
- Misra J, Gries D (1982) Finding repeated elements. *Sci Comput Program* 2(2):143–152

- Nasir MAU, Morales GDF, García-Soriano D, Kourtellis N, Serafini M (2015) The power of both choices: Practical load balancing for distributed stream processing engines. In: 31st IEEE international conference on data engineering, ICDE 2015, Seoul, 13–17 Apr 2015, pp 137–148
- Olston C, Jiang J, Widom J (2003) Adaptive filters for continuous queries over distributed data streams. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, 9–12 June 2003, pp 563–574
- Stonebraker M, Çetintemel U, Zdonik SB (2005) The 8 requirements of real-time stream processing. SIGMOD Rec 34(4):42–47
- Tatbul N, Çetintemel U, Zdonik SB, Cherniack M, Stonebraker M (2003) Load shedding in a data stream manager. In: VLDB 2003, Proceedings of 29th international conference on very large data bases, 9–12 Sept 2003, Berlin, pp 309–320
- Teubner J, Müller R (2011) How soccer players would do stream joins. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2011, Athens, 12–16 June 2011, pp 625–636
- Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57
- Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: ACM SIGOPS

U

Uncertain Data Integration

- ▶ Probabilistic Data Integration

Definitions

A set of methods to support a matching process among elements of multiple data sources, using uncertainty management tools to quantify the inherent uncertainty in the process.

Uncertain Ontology Alignment

- ▶ Uncertain Schema Matching

Overview

Data integration has been the focus of research for many years now, motivated by data heterogeneity (arriving from multiple sources), the lack of sufficient semantics to fully understand the meaning of data, and errors that may stem from incorrect data insertion and modifications (e.g., typos and eliminations).

At the heart of the data integration process is a schema matching problem (Bellahsene 2011) whose outcome is a collection of correspondences between different representations of the same real-world construct. Schema matching research has been going on for more than 25 years now, first as part of schema integration and then as a stand-alone research field (see surveys Batini et al. 1986; Rahm and Bernstein 2001; Shvaiko and Euzenat 2005). Over the years, a significant body of work has been devoted to the identification of *schema matchers*, heuristics for schema matching. The main objective of schema matchers is to provide correspondences. Examples of algorithmic tools used for schema matching include COMA (Do and Rahm 2002),

Uncertain Schema Mapping

- ▶ Uncertain Schema Matching

Uncertain Schema Matching

Avigdor Gal
Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel

Synonyms

Uncertain ontology alignment; Uncertain schema mapping

Cupid (Madhavan et al. 2001), OntoBuilder (Modica et al. 2001), Autoplex (Berlin and Motro 2001), Similarity Flooding (Melnik et al. 2003), Clio (Miller et al. 2001), Glue (Doan et al. 2002), and others (Bergamaschi et al. 2001; Saleem et al. 2007). These have come out of a variety of different research communities, including database management, information retrieval, the information sciences, data semantics and the semantic Web, and others. Research papers from different communities have yielded overlapping, similar, and sometimes identical results. Meanwhile, benchmarks, such as the OAEI ([Ontology alignment evaluation initiative](#)), indicate that the performance of schema matchers still leaves something to be desired.

In recent years, data integration has been facing new challenges as a result of the evolution of data accumulation, management, analytics, and visualization (a phenomenon known as *big data*). Big data is mainly about collecting *volumes* of data in an increased *velocity* from a *variety* of sources, each with its own level of *veracity*. Big data encompasses technological advancements such as the Internet of things (accumulation), cloud computing (management), and deep learning (analytics), packaging it all together while providing an exciting arena for new and challenging research agenda. The veracity of data puts aspects of uncertainty as a forefront challenge in any big data integration task. In particular, a matcher may consider several possible correspondences as possible, and when it needs to choose, it may choose wrong (Gal 2006). For example, when two companies merge, the companies' employee databases need to be consolidated. The schemata may be different, and in such a case, there may be many different ways of mapping one schema to another. As another example, a Web search engine performing a product search over databases of multiple vendors needs to find correspondences between the product databases of different vendors. Multiple ways of representing the data might lead to multiple possible correspondences.

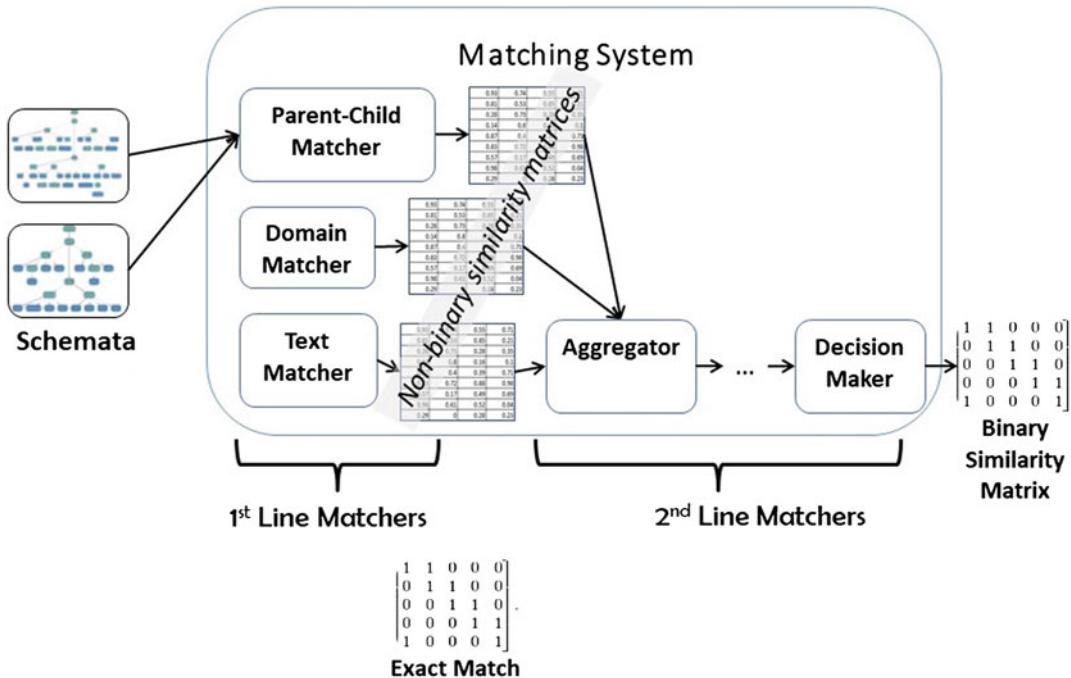
The realization that schema matchers are inherently uncertain has emerged even before the age of big data. A prime reason for the

uncertainty of the matching process is the enormous ambiguity and heterogeneity of data description concepts: it is unrealistic to expect a single matcher to identify the correct mapping for any possible concept in a set. Madhavan et al. (2002) argued for the need “to incorporate *inaccurate* mappings and handle *uncertainty* about mappings. Inaccuracy arises because in many contexts there is no precise mapping... mappings may be inaccurate [since] the mappings language is too restricted to express more accurate mappings.” Since 2003, work on the uncertainty in schema matching has picked up (along with research on uncertainty in other areas of data management) (Dong et al. 2009; Gal et al. 2009).

Key Research Findings

A matching process matches two schemata by aligning their attributes. Many matching techniques exist, see surveys (e.g., Bernstein et al. 2011) and books (e.g., Gal 2011), using matching cues such as attribute names, instance data, schema structure, etc. The result of a matching problem is a collection of correspondences between attributes. This collection is often formalized as a similarity matrix between the corresponding schemata. Each row and column in the matrix correspond to one of the matched attribute sets. Therefore, each entry in the matrix represents the degree of confidence that the two attributes correspond. Confidence is usually expressed as a real value in $[0, 1]$. Binary similarity matrices are constrained to allow entries in $\{0, 1\}$ only (match/non-match).

Matching is often a stepped process (see Fig. 1) in which different algorithms, rules, and constraints are applied. Several classifications of matching steps have been proposed over the years. Matchers can be separated into those that are applied directly to the problem (*first-line matchers* – 1LMs) and those that are applied to the outcome of other matchers (*second-line matchers* – 2LMs). 1LMs receive two schemata and return a similarity matrix. Table 1 provides an example of the outcome of a 1LM, matching two



Uncertain Schema Matching, Fig. 1 A matching system illustrated

Uncertain Schema Matching, Table 1 Non-binary similarity matrix

$S_1 \rightarrow$ $\downarrow S_2$	cardNum	city	arrival Day	checkIn Time
clientNum	0.84	0.32	0.32	0.30
city	0.29	1.00	0.33	0.30
checkInDate	0.34	0.33	0.35	0.64

Uncertain Schema Matching, Table 2 Binary similarity matrix

$S_1 \rightarrow$ $\downarrow S_2$	cardNum	city	arrival Day	checkIn Time
clientNum	1	0	0	0
city	0	1	0	0
checkInDate	0	0	0	1

schemata in a reservation system. Such a matcher gives high scores to attributes with similar names. 2LMs receive one or more similarity matrices and return a similarity matrix. Among the 2LMs, we term *decision-makers* those that return a binary matrix as an output, also known as *alignments as solutions* (Shvaiko and Euzenat 2005). For example, Table 2 represents the outcome of

a decision-making 2LM that enforces a 1 : 1 matching over the similarity matrix of Table 1.

ILMs are bound to provide uncertain results. Similarity functions return an uncertain value, typically between 0 and 1. Most data integration evaluation methods to date use a binary (match/no match) approach, while the variety and veracity of big data require to broaden evaluation to a full scope in between. As a concrete example, consider the evaluation measures of precision and recall, common in the area of information retrieval. These measures test the completeness and soundness of a matcher outcome with respect to some reference match. Precision and recall are based on a binary correspondence comparison, which requires binary decision-making from the matcher side and a binary reference match. With big data integration, matchers may not be able to provide a crisp decision regarding a correspondence and may prefer to provide an uncertain one (in terms, e.g., of a probability). On the other hand, research shows (Sagi and Gal 2014) that experts may fail to agree on a reference match and instead multiple opinions can be gathered regarding the correctness of a correspondence.

Utilizing existing measures requires us to binarize both algorithmic and human aggregate responses and implicitly assume that the human’s result is to be preferred over the algorithmic response. Since we do not assume humans to be inherently better at matching, especially if they are given partial context as part of a pay-as-you-go scheme (e.g., Zhang et al. 2013), information regarding both parties’ uncertainty should be retained and used in disagreement resolution. An initial work on non-binary evaluation, using finite real vector space representation, extends traditional measures and suggests a fresh approach toward quality measures in an uncertain setting (Sagi and Gal 2018).

Many of the 2LMs are decision-makers, which is modeled as a binary similarity matrix. When it comes to uncertain schema matching, one may take advantage of an explicit modeling of uncertainty as part of the schema matching process; see, for example, the use of Bayesian probabilistic models for matching (Assoudi and Lounis 2015).

Madhavan et al. (2002) argue that “when no accurate mapping exists, the issue becomes one of choosing the *best* mapping from the viable ones.” The latter approach was later extended to handle Top- K multiple schema matches in parallel (Gal 2006; Domshlak et al. 2007). Top- K schema matches are intuitively defined as a ranked list of the best K schema matches a matcher can generate. The formal definition is recursive, providing interesting insights into the behavior patterns of matchers. Top- K schema matches play a pivotal role in managing uncertain schema matching. The effectively unlimited heterogeneity and ambiguity of data description suggest that in many cases an exact match will not be identified as a best match by any schema matcher. Therefore, top- K schema matches serve to create a search space in uncertain settings (Bernstein et al. 2006) and can also help improve the precision of matching results (Gal 2006). All in all, empirical results have shown the top- K list to be a quality set of candidates for schema matching.

Extended solutions, which consist of multiple possible matches, lead naturally to *probabilistic schema matching*, methods that quantify the rela-

tive confidence in multiple match results and use this quantification for query answering. Probabilistic schema matching studies two semantics for answering queries over multiple databases using probabilistic schema matches – a *by-table* semantics and a *by-tuple* semantics (Dong et al. 2009). The former semantics represents a situation where a single match should be applied to the entire set of tuples in the source relation. As for the latter, a choice of a match should be made for each of these tuples independently. The *by-tuple* semantics represents a situation in which data is gathered from multiple sources, each with a potentially different interpretation of a schema. An extension to the basic SPJ query answering under probabilistic schema matching into aggregations was presented by Gal et al. (2009).

An interesting aspect of uncertain schema matching is the possible lack of a reference match. Reference matches, the outcome of a manual validation, are rare, hard to come by and generate. In the absence of a reference match, schema matchers perform a “best effort” matching without any indication of the prospective success of their efforts. These observations led to research of *schema matching prediction* (Tu and Yu 2005; Sagi and Gal 2013), an assessment mechanism to support schema matchers in the absence of a reference match. A predictor measures the success of a matcher in identifying correct correspondences based on the matcher’s pair-wise similarity measures. For a predictor to be useful, its predictions should correlate well with the quality of the matcher’s decision-making.

Examples of Application

Over the years, schema matching research has expanded and specialized to answer research and application questions in a variety of domains. In its origin, schema matching was conceived to be a preliminary process to schema mapping to serve applications such as query rewriting and database integration. An underlying basic assumption of this original thinking was that schema matching

provides a set of definite (true or false) correspondences to be then validated by some human expert before mapping expressions are generated. This is no longer a valid assumption when it comes to uncertain schema matching. For example, consider an enterprise that is interested in investigating a new market, seeking partners with sufficient common interests to its own while providing sufficient added value to its capabilities. Matching here can ensure just the right amount of commonality between prospective business partners, judging what part of one schema can be covered (Saha et al. 2010) by attributes of another schema and vice versa. Also, effective reuse of previous matching results requires the accumulation of many attributes from which useful schemata for a specific need can be created. To ensure efficient processing, attribute repositories can be used to filter that only attributes of good quality are retrieved.

Prediction offers a promise for a whole new set of applications by presenting a designer with a quality assessment of a match as a first step toward decision-making under uncertainty. Prediction can serve in directing the flow of the integration process. For example, it can serve as a decision factor on whether the integration of two databases should be preferred over the design from scratch of a new database application. Prediction can also serve other schema matching tasks such as matcher self-configuration, matcher selection, user involvement optimization, etc. One example that illustrates the usefulness of prediction involves schema matcher ensemble construction. Contemporary ensemble tools, e.g., LSD (Doan et al. 2001), perform offline learning of matcher weights in an ensemble. Dynamically assigning weights can be done based on the matching outcome of individual matchers and their success prediction.

Future Directions for Research

The research area of schema matching is always in need of new heuristics, ILMs, to be added to ensembles of matchers and enable a better auto-

matic matching. Uncertainty management should be part of any such new heuristic to allow a better control over the quality of the matching outcome.

The ability to tap into crowdsourced and expert opinions in a piecemeal fashion has allowed pay-as-you-go frameworks for data integration (e.g., Hung et al. 2013; Zhang et al. 2013), to make flexible use of human input in integration processes. Such methods of evaluation should be carefully monitored from a perspective of uncertainty management.

References

- Assoudi H, Lounis H (2015) Coping with uncertainty in schema matching: Bayesian networks and agent-based modeling approach. Springer International Publishing, Cham, pp 53–67
- Batini C, Lenzerini M, Navathe S (1986) A comparative analysis of methodologies for database schema integration. ACM Comput Surv 18(4):323–364
- Bellahsene Z (2011) Schema matching and mapping. Springer, New York
- Bergamaschi S, Castano S, Vincini M, Beneventano D (2001) Semantic integration of heterogeneous information sources. Data Knowl Eng 36(3):215–249
- Berlin J, Motro A (2001) Autoplex: automated discovery of content for virtual databases. Springer, London, pp 108–122
- Bernstein P, Melnik S, Churchill J (2006) Incremental schema matching. Proc Int Conf Very Large Databases 2:1167–1170
- Bernstein P, Madhavan J, Rahm E (2011) Generic schema matching, ten years later. PVLDB 4(11):695–701
- Do H, Rahm E (2002) COMA – a system for flexible combination of schema matching approaches. In: Proceedings of the 8th international conference on very data bases, pp 610–621
- Doan A, Domingos P, Halevy A (2001) Reconciling schemas of disparate data sources: a machine-learning approach. In: Proceedings of the ACM SIGMOD international conference on management of data, pp 509–520
- Doan A, Madhavan J, Domingos P, Halevy A (2002) Learning to map between ontologies on the semantic web. In: Proceedings of the 11th international world wide web conference, pp 662–673
- Domshlak C, Gal A, Roitman H (2007) Rank aggregation for automatic schema matching. IEEE Trans Knowl Data Eng (TKDE) 19(4):538–553
- Dong X, Halevy A, Yu C (2009) Data integration with uncertainty. VLDB J 18:469–500
- Gal A (2006) Managing uncertainty in schema matching with top-k schema mappings. J Data Semant 6:90–114

- Gal A (2011) Uncertain schema matching. *Synthesis lectures on data management*. Morgan & Claypool Publishers, San Rafael
- Gal A, Martinez M, Simari G, Subrahmanian V (2009) Aggregate query answering under uncertain schema mappings, pp 940–951
- Hung N, Tam N, Miklós Z, Aberer K (2013) On leveraging crowdsourcing techniques for schema matching networks. In: Meng W, Feng L, Bressan S, Winiwarter W, Song W (eds) *Database systems for advanced applications. Lecture notes in computer science*, vol 7826. Springer, Berlin/Heidelberg, pp 139–154
- Madhavan J, Bernstein P, Rahm E (2001) Generic schema matching with Cupid, pp 49–58
- Madhavan J, Bernstein P, Domingos P, Halevy A (2002) Representing and reasoning about mappings between domain models, pp 80–86
- Melnik S, Rahm E, Bernstein P (2003) Rondo: a programming platform for generic model management. In: *Proceedings of the ACM SIGMOD international conference on management of data*, pp 193–204
- Miller R, Hernández, M, Haas L, Yan LL, Ho C, Fagin R, Popa L (2001) The Clio project: managing heterogeneity. *SIGMOD Rec* 30(1):78–83
- Modica G, Gal A, Jamil H (2001) The use of machine-generated ontologies in dynamic information seeking. In: *Proceedings of the international conference on cooperative information*, pp 433–448
- Ontology alignment evaluation initiative. <http://oaei.ontologymatching.org/>
- Rahm E, Bernstein P (2001) A survey of approaches to automatic schema matching. *VLDB J* 10(4):334–350
- Sagi T, Gal A (2013) Schema matching prediction with applications to data source discovery and dynamic ensembling. *VLDB J* 22(5):689–710
- Sagi T, Gal A (2014) In schema matching, even experts are human: towards expert sourcing in schema matching. In: *Workshops proceedings of the 30th international conference on data engineering workshops, ICDE 2014, Chicago, 31 Mar–4 Apr 2014*, pp 45–49
- Sagi T, Gal A (2018) Non-binary evaluation measures for big data integration. *VLDB J* 27(1):105–126
- Saha B, Stanoi I, Clarkson K (2010) Schema covering: a step towards enabling reuse in information integration. In: *Proceedings of the 26th international conference on data engineering*, pp 285–296
- Saleem K, Bellahsene Z, Hunt E (2007) Performance oriented schema matching. In: *Proceedings of the 18th international conference database and expert systems application*, pp 844–853
- Shvaiko P, Euzenat J (2005) A survey of schema-based matching approaches. *J Data Semant* 4:146–171
- Tu K, Yu Y (2005) CMC: combining multiple schema-matching strategies based on credibility prediction. In: Zhou L, Ooi B, Meng X (eds) *Database systems for advanced applications. Lecture notes in computer science*, vol 3453. Springer, Berlin/Heidelberg, pp 995–995
- Zhang JC, Chen L, Jagadish H, Cao CC (2013) Reducing uncertainty of schema matching via crowdsourcing. *Proc VLDB Endow* 6:757–768

Uncertainty in Streams

Avigdor Gal and Nicoló Rivetti
Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel

Definitions

Models and algorithms that support a stream of events that may demonstrate uncertainty in event occurrence, as well as in values assigned to events in a stream.

Overview

Many contemporary applications depend on the ability to monitor efficiently streams of events (e.g., application messages or business events) to detect and react in a timely manner to situations. Some events are generated exogenously by devices such as sensors and flow across distributed systems. Other events (and their content) are inferred by complex event processing (CEP) systems. The first generation of CEP systems was built as stand-alone prototypes or as extensions of existing database engines. These systems were diversified into products with various approaches toward event processing, including stream-oriented, rule-oriented, imperative, and publish-subscribe paradigms. Common to all of these approaches is the assumption that received events have occurred and that the CEP system is complete. In other words the system captures all events and the values carried by events are deterministic and correct as the processing of the event itself. Second-generation CEP systems were extended to handle fault tolerance, adaptive query processing, and enhanced expressiveness. This chapter focuses on a specific functional aspect, namely, the ability to deal with uncertainty in event streams.

There are four basic approaches to handle uncertainty in event streams. First, in applications where uncertain situation are infrequent, simply

ignoring uncertainty turns out to be cost-effective since it may induce only negligible harm to the overall analysis. Alternatively, a deterministic approach may require absolute evidence to detect the situation. A third way to handle uncertainty is to get assistance from additional sources in case a situation detection is unreliable. A common solution (e.g., fraud detection) is to introduce a human observer in the loop, which monitors the detected situations and filters out false positives. Finally, a fourth approach is to quantify uncertainty (e.g., using probability theory) and then use tools such as utility theory to assist in decision-making.

Use-cases for second generation CEP engines include applications where the rate of false positives and negatives is non negligible, and simplistic approach to uncertainty may induce relevant, if not critical, damage. While the motivation exists, in general the handling of uncertainty in CEP is still a challenge. A few solutions have been presented, some being application dependent. Wasserkrug et al. (2012a,b) have presented the first model that deals with uncertainty in CEP. The model assumes independence of simple event occurrence, as in Zhang et al. (2013). Cugola et al. (2015) extended it also to complex events, while Ré et al. (2008) apply the Markovian hypothesis. It is worth noting that of all these works, only in Zhang et al. (2013) temporal uncertainty, is considered. These works leverage probability theory and Bayesian networks to build a probabilistic model, which in turn propagates the uncertainty from simple to complex events. In particular, Cugola et al. (2015) also take into account uncertainty at the rule level (i.e., incomplete or wrong assumptions about the environment).

Event Processing — An *event* is an occurrence within a particular system or domain (Etzion and Niblett 2010). It also represents the programming entity of the occurrence itself in a CEP engine. An event can either be *raw* or *derived*. A raw event is an event that is produced by an event producer (source), while a derived event is an event that is generated through the processing performed by the CEP engine. A derived event made of the composition of multiple events is said to

be composite. An event *pattern* is a template, specifying one or more relevant combinations of events. The matching of the pattern against the stream generates a *pattern matching set*, i.e., the set of sequences of events satisfying the pattern conditions. The pattern matching set can, in turn, become a derived event, either as a composite event or as an event resulting from some transformation applied to the set. A *situation* identifies an event occurrence (either raw or derived) that has to be handled.

Uncertainty Models

This section briefly describes three of the main alternatives for modeling uncertainty: *probability theory*, *fuzzy sets* (Zadeh 1965), and *possibility theory* (Halpern 2003). The three models differ in the way uncertainty is conceived and measured.

The **Probability theory** assumes that imperfect knowledge can be encoded as probabilities about sets of possible worlds. An intuitive way to define a probability space involves possible world semantics (Green and Tannen 2006). A probability space is a measure space with total measure one and is defined by a triple (W, F, μ) such that:

- W is the set of possible worlds, with each corresponding to a specific set of event occurrences that is considered possible.
- $F \subset 2^{|W|}$ is an algebra over W and F is a non-empty collection of sets of possible worlds that is closed under countable union and complementation.
- $\mu : F \rightarrow [0, 1]$ is a probability measure over F .

U

As an example, consider an unbiased coin single flip which can either be head or tail. We then have, i.e., $W = \{h, t\}$ while the algebra is $F = \{\emptyset, \{h\}, \{t\}, \{h, t\}\}$, where \emptyset is neither head nor tail and $\{h, t\}$ is head or tail. Finally, the probability measure μ is such that $\mu(\emptyset) = 0$, $\mu(\{h\}) = 0.5$, $\mu(\{t\}) = 0.5$, $\mu(\{h, t\}) = 1$

Bayesian (or belief) networks (Pearl 1988) are the most widely accepted paradigm for inference under uncertainty. Probability theory is used to capture the uncertainty by representing each

conjecture with a random variable. The random variables are modeled as nodes in a graph, where edges represent conditional dependencies. Such a graph, augmented by conditional probability tables, is a qualitative representation of the dependencies between nodes. Each node is assigned with the probability of each of its values, given the values of its parents. Several inference algorithms exist based upon this augmented graphical structure. To model temporal aspects, multiple extensions of Bayesian networks have been defined including Time Nets (Kanazawa 1991) and Temporal Nodes Bayesian Networks (Arroyo-Figueroa and Sucar 1999).

Probabilistic logics (Bacchus 1990; Halpern 1990) are general formal models suitable for the representation of probabilistic rules and may therefore be used to provide a formal setting for the representation of probabilistic knowledge in any particular domain. However, such logics are less suitable than Bayesian networks for the inference of probabilities given evidence.

A third alternative is the knowledge-based model construction (KBMC) paradigm. KBMC contains any probabilistic system in which the representation and inference models are separated. However, in most existing applications and specializations of this approach, knowledge is represented in a probabilistic logic, and inference is carried out by transforming the logical statements into a Bayesian network.

Fuzzy set theory defines a membership of an element x in a fuzzy set M on a universe U using a membership measure $\mu_M : U \rightarrow [0, 1]$, indicating the extent in which it is associated with the set (Zadeh 1965; Gal et al. 2005; Liu and Jacobsen 2004). Unlike probability theory, the degree of membership of an element over all fuzzy sets M does not necessarily sum to 1. Fuzzy logics are methods for reasoning with logical expressions describing membership in fuzzy sets. These logics define operators, such as triangular norms and fuzzy aggregates, to compute the membership measure of complex events.

Possibility theory associate a confidence measure π to an event (Dubois and Prade 1988). The possibility measure Π expresses the likelihood of event e occurrence, while the necessity measure N represents the unlikelihood

of the complement of e (Liu and Jacobsen 2004). These two measures satisfy the following equations:

$$N(e) = 1 - \Pi(\bar{e}) \quad (1)$$

$$\forall e, \Pi(e) \geq N(e) \quad (2)$$

If an event certainly occurs, then $\Pi(e) = 1$, and if it cannot occur, then $\Pi(e) = 0$. The intermediate values in $[0, 1]$ express different degrees of confidence in the event occurrence. Notice that, if $\Pi(e)$ is a probability distribution, then $\Pi(e) = N(e)$.

Discussion — Probabilistic and fuzzy set theory are competing methods for modeling uncertainty. A probabilistic approach assumes that there is no complete knowledge on what is modeled. However, this knowledge can be encoded as probabilities on events. The fuzzy approach aims at modeling the intrinsic imprecision of attributes of the modeled reality. Probabilistic reasoning relies on event independence assumptions, making event correlation hard or even impossible to evaluate. A study (Drakopoulos 1994) shows that probabilistic analysis is more expressive than fuzzy sets; however fuzzy sets are computationally more efficient.

Both fuzzy set theory and possibility theory use a numerical measure; however the former is more suited to represent uncertain description of an event (e.g., temperature value), while the latter defines the confidence in the occurrence of the event.

Element and Origin Uncertainty

Events uncertainty can be classified according to two orthogonal dimensions, namely, *element uncertainty* and *origin uncertainty*. The first dimension, element uncertainty, either refers to the uncertainty of the event occurrence or its attribute values. Uncertainty regarding event occurrence comes from the fact that the system does not know whether or not this event has in fact occurred. The uncertainty concerning the event attributes stems from inaccuracies in reported values, e.g., granularity, skew, or drift in clocks.

The second dimension, origin uncertainty, relates to both raw and derived events. The uncer-

tainty can thus originate at the source itself or can be traced to the uncertain inference performed by the systems on other events.

Due to the orthogonality of these dimensions, four types of event uncertainty can be identified:

- Uncertainty regarding event occurrence originating at an event source
- Uncertainty regarding event occurrence resulting from inference
- Uncertainty regarding event attributes originating at an event source
- Uncertainty regarding event attributes resulting from event inference

Causes of Source and Inference Uncertainty

Source uncertainty — Uncertainty regarding event occurrence at the source is caused by one of the following:

- An event source may malfunction and indicate that an event has occurred even if it has not or the converse.
- An event source may fail to signal the occurrence of events due to limited precision or may signal events that did not occur.
- The communication medium between the source and the CEP system may drop or intrude spurious events.
- In some cases, the source itself may produce estimations as events.

Uncertainty regarding the attributes originating at the event source can also be caused by any of the above reasons. Considering distributed systems, the skew and drift among the clocks of the involved entities introduce an additional source of uncertainty.

Note that in both of the above cases, uncertainty regarding a specific event may be caused by a combination of factors. For example, it is possible that both the event source itself and the communication medium simultaneously corrupt the same event.

Inference uncertainty — The uncertainty regarding event occurrence resulting from inference has the two possible causes: *propagation of uncertainty* or *uncertain rules*. While an inferred event can be a result of a deterministic rule, if the inference is made over uncertain events, then their uncertainty propagates to the inferred events. On the other hand, the rules may not be deterministic themselves. For instance, deterministic methods to identify malicious behavior may induce false negatives as well as false positives.

Source and inference uncertainty may be combined. That is, it may happen that not only is the inference of an event based on an uncertain rule, but also uncertainty exists regarding the occurrence (or attribute values) of one of the events which serves as evidence for this inference.

Considering uncertainty of inferred event attributes, the possible causes depend on how these attributes are calculated from the attributes of the evidence events. The most intuitive way to calculate such inferred attributes is by using deterministic functions defined over the attributes of the evidence events. In such a case, the only cause of uncertainty in the inferred attributes is rooted in the uncertainty of attribute values of the evidence events. Therefore, the rule cannot induce uncertainty regarding the attribute values of the inferred events. However, if the inference system makes it possible to define the attribute values of the inferred events in a nondeterministic manner, uncertain rules may also be a cause of inferred attribute uncertainty.

Key Research Findings

Handling Uncertainty through Probability Theory

Modeling uncertain raw events — The first model represents the uncertainty associated with event through probability theory. Such choice is supported by the following reasons:

- Probability theory has widespread acceptance, is well-understood, and is supported by powerful tools.

- Its use is facilitated by many technical results supported by formal justification.
- Under certain assumptions, probability is the only “rational” way to represent uncertainty (Halpern 2003).
- There are well-known and accepted methodologies for carrying out inferences based on probability theory, involving structures known as Bayesian networks (Pearl 1988).
- Probability theory can be used together with utility theory (Raiffa 1997) for automatic decision-making. Such automatic decision-making would facilitate the implementation of automatic actions by CEP engines.

The CEP engine maintains for each event a data structure dubbed Event Instance Data (EID). An EID incorporates all relevant information about an event, such as its type and time of occurrence. An event with n attributes (e.g., type, timestamp, etc) can be represented as a tuple of values $t = \langle v_1, \dots, v_n \rangle$, one value for each attribute. To capture the uncertainty on an event instance, the EID is a random variable with domain $\mathcal{V} = V \cup \{\perp\}$, where V is a set of tuples $\langle v_1, \dots, v_n \rangle$, each representing a possible combination of the event attributes values. A probability can then be attached to any subset of $S \subset \mathcal{V}$, expressing the probability that the event attribute values match one of the possible tuples in S , while associated with \perp is the probability of nonoccurrence of event e . This representation captures both the uncertainty regarding the attributes and occurrence. An additional concept, relevant in the context of event composition systems, is that of event history (EH). An event history $EH_{\tau_1}^{\tau_2}$ is the set of all events, as well as their associated data, whose occurrence time falls between τ_1 and τ_2 .

Complex event patterns — Deterministic CEP engines use pattern matching for situation detection. In such systems, a situation is said to have occurred if the stream of incoming events match some pattern. The event pattern notion can be extended to represent the uncertainty associated with event derivation (Wasserkrug et al. 2012a). This extension is based on

probability theory; however the framework can also be applied to fuzzy set theory or possibility theory.

At each point in time τ , the existence of relevant patterns can be checked in the event histories that are known at that time. A pattern p is defined as a tuple $\langle sel_p^n, pred_p^n, type_p, mapping_p, prob_p \rangle$ where:

- sel_p^n is a deterministic predicate returning a subset of an event history of size less than or equal to n (for some integer n). If the returned subset has strictly less than n elements, no further evaluation of the rule is carried out.
- $pred_p^n$ is a (possibly complex temporal) predicate of arity n over event instances (this is the same n appearing in sel_p^n). This predicate is applied to the n event instances selected by sel_p^n .
- $type_p$ is the type of event inferred by this pattern.
- $mapping_p$ is a set of functions mapping the attribute values of the events that matched this pattern to the attribute values of the inferred event.
- $prob_p \in [0, 1]$ is the probability of inferring the event by the pattern.

The quantity $prob_p$ is the only uncertainty associated with the pattern.

Inference over uncertain events — Pattern reasoning engines must be able to compute at any time τ the probability that an event has occurred at time $\tau' \leq \tau$, based only on the knowledge available to the engine at time τ . Therefore, a (possibly different) probability space (cf., section “[Uncertainty Models](#)”) is defined for each τ , as a tuple $P_\tau = (W_\tau, F_\tau, \mu_\tau)$. W_τ is a set of possible worlds, with each possible world corresponding to a specific finite event history that is considered possible at time τ . The event history, as well as the overall number of events, is assumed to be finite since a system cannot consider an infinite number of events.

A computationally more efficient, while less intuitive, probability space definition is the following. Let E_1, E_2, \dots be the set of EIDs repre-

senting the information about all events of interest. Each finite event history can be represented by a finite number of values e_1, \dots, e_n , such that there is a finite number of EIDs E_1, \dots, E_n where e_i is a possible value of E_i . Then, each possible world $w_\tau \in W_\tau$ can be represented by such a finite number of values. Moreover, since the overall number of events is finite, there is a finite number of events E_1, \dots, E_m such that E_i has occurred in some $w_\tau \in W_\tau$. Finally, if $|W_\tau|$ is finite, each E_i can only have a finite number of associated values (one for each world in W_τ) in which it appears. Note that in such a case, each possible world w_τ can be represented by a finite number of values t_1, \dots, t_m , where the values t_1, \dots, t_n for some $n \leq m$ represent the values of one of the n events that occurred in w_τ , and t_{n+1}, \dots, t_m are all $\{\perp\}$. It follows that if the probability space P_τ represents the knowledge of the composite event system at time τ , this knowledge can be represented by a set of m EIDs - E_1, \dots, E_m .

Therefore, when $|W_\tau|$ is finite, it is possible to define the probability space P_τ as $(\Omega_\tau, F_\tau, \mu'_\tau)$ where:

- $\Omega_\tau = \{t_1, \dots, t_m\}$ such that the tuple t_1, \dots, t_m is the set of values corresponding to an event history, where this event history is a possible world $w_\tau \in W_\tau$ as described above. Obviously, $|\Omega_\tau|$ is finite.
- $F_\tau = 2^{|\Omega_\tau|}$
- $\mu'_\tau(\{t_1, \dots, t_m\}) = \mu_\tau(w_\tau)$ such that w_τ is the world represented by $\{t_1, \dots, t_m\}$

This representation is called the *EID representation*.

Intuitively, the semantics of each pattern p is as follows: Let $EH_{\tau_1}^{\tau_2}$ be an event history. If the pattern p is applied at some time $\tau \geq \tau_2$ and the set of events selected by sel_p^n from $EH_{\tau_1}^{\tau_2}$ is of size n and is such that $pred_p^n$ on this event is true, then the event inferred by rule p occurred with probability $prob_p$. The value of its corresponding attributes is the value defined by $mapping_p$. Otherwise, the event cannot be inferred.

Formally, let $sel_p^n(EH_{\tau_1}^{\tau_2})$ denote the set of events selected by sel_p^n from $EH_{\tau_1}^{\tau_2}$, and let $pred_p^n(sel_p^n(EH_{\tau_1}^{\tau_2}))$ denote the value of the predicate $pred_p^n$ on $sel_p^n(EH_{\tau_1}^{\tau_2})$ (recall that if $|sel_p^n(EH_{\tau_1}^{\tau_2})| < n$ then the rule is not applied). In addition, let v_1, \dots, v_n denote the value of the attributes of the inferred event e_p as defined by $mappingExpressions_p$. Then, if the specific event history is known and denoting by E_p the EID corresponding to e_p , we have the following:

$$\begin{aligned} \mathbb{P}(E_p = \{v_1, \dots, v_n\} | EH_{\tau_1}^{\tau_2}) &= prob_p \\ \text{if } pred_p^n(sel_p^n(EH_{\tau_1}^{\tau_2})) &= true \end{aligned}$$

$$\begin{aligned} \mathbb{P}(E_p = \{\perp\} | EH_{\tau_1}^{\tau_2}) &= (1 - prob_p) \\ \text{if } pred_p^n(sel_p^n(EH_{\tau_1}^{\tau_2})) &= true \end{aligned}$$

$$\begin{aligned} \mathbb{P}(E_p = \{\perp\} | EH_{\tau_1}^{\tau_2}) &= 1 \\ \text{if } pred_p^n(sel_p^n(EH_{\tau_1}^{\tau_2})) &= false \end{aligned}$$

For a finite W_τ , the probability space can be represented by a finite set of finite random variables. In addition, note that the rule semantics defined above specify that the probability of the inferred event does not depend on the entire event history but rather on the events selected by sel_p^n . Let us denote by E_1, \dots, E_m the set of EIDs that describe knowledge regarding the event history and let $\{E_{i_1}, \dots, E_{i_l}\}$ describe the subset of $\{E_1, \dots, E_m\}$ that are candidates for selection by sel_p^n (note that $l \geq n$, as sel_p^n must choose the first n events that have **actually occurred**). An EID E is a candidate for selection if there is a possible event history in the probability space P_τ such that there is a set of n events, chosen by sel_p^n from this event history, and the event e corresponding to E is in this set. Then, for all sets of values $\{t_1, \dots, t_m\}$ such that $E_i = t_i$, we have that:

$$\mathbb{P}(E_p | E_{i_1}, \dots, E_{i_l}) = \mathbb{P}(E_p | E_1, \dots, E_m) \quad (3)$$

i.e., E_p is conditionally independent of $\{E_1, \dots, E_m\} \setminus \{E_{i_1}, \dots, E_{i_l}\}$ given $\{E_{i_1}, \dots, E_{i_l}\}$. Now let t_{i_1}, \dots, t_{i_m} denote a specific set of values of

E_{i_1}, \dots, E_{i_l} . Given such a set of specific values, the subset $\{e'_{j_1}, \dots, e'_{j_n}\}$ selected by sel_p^n is well

defined. Therefore, we have from the above equations that:

$$\mathbb{P}(E_p = \{v_1, \dots, v_n\} | t_{i_1}, \dots, t_{i_m}) = prob_p \text{ if } pred_p^n(e'_{j_1}, \dots, e'_{j_n}) = true \quad (4)$$

$$\mathbb{P}(E_p = \{\perp\} | t_{i_1}, \dots, t_{i_m}) = (1 - prob_p) \text{ if } pred_p^n(e'_{j_1}, \dots, e'_{j_n}) = true \quad (5)$$

$$\mathbb{P}(E_p = \{\perp\} | e_{i_1}, \dots, e_{i_m}) = 1 \text{ if } pred_p^n(e'_{j_1}, \dots, e'_{j_n}) = false \quad (6)$$

Inference algorithm — We now detail an algorithm for uncertain inferencing of events (Wasserkrug et al. 2008). In a nutshell, the algorithm works as follows: Given a set of patterns and a set of EIDs at time τ , a Bayesian network is automatically constructed, representing the probability space at τ according to the semantics previously defined. Probabilities of new events are then computed using standard Bayesian network methods.

Any algorithm for event derivation must support both termination and determinism (Widom and Ceri 1994). The former guarantees that the algorithm will terminate. Determinism ensures that for the same input (i.e., set of explicit EIDs), the derivation result will be always the same. The Bayesian network construction algorithm should provide two additional features, namely, efficiency and dynamic updates. The latter determines that the network represents the probability space P_τ at all times. Event derivation algorithms are likely to store some additional meta data to support efficiency.

Patterns are assumed to be acyclic, and, to enforce determinism in the activation order, each is assigned a priority. By definition, the occurrence time of each inferred event is the time in which the pattern was applied. Therefore, the single possible occurrence time $E.\sigma\tau$ of each EID defines a full order on the EIDs. According to Eq. 3, each EID is independent of all preceding EIDs, given the EIDs that may be selected by the selection expression. Therefore, we construct a Bayesian network such that the nodes of the network consist of the set of random variables in the system event history, and an edge exists between EID E_1 and EID E_2 iff $E_1.\sigma\tau \leq E_2.\sigma\tau$

and E_2 is an EID corresponding to an event that may be inferred by pattern p , where the event corresponding to E_1 may be selected by sel_p^n . A network constructed by these principles encodes the probabilistic independence required (Pearl 1988) by Eq. 3. This structure is now augmented with values based on Eqs. 4, 5, and 6.

Based on the above principles, a Bayesian network is constructed and dynamically updated as events enter the system. At each point in time, nodes and edges may be added to the Bayesian network. The algorithm below describes this dynamic construction. The information regarding the new event is represented by some EID E , the system event history is represented by EH , and the constructed Bayesian network by BN .

1. $EH \leftarrow EH \cup \{E\}$
2. Add a node for E to BN .
3. For each such pattern p in a decreasing priority order:
 - a. Denote by $sel_p^n(EH)$ the subset of EIDs in EH that may be selected by sel_p^n (these are all EIDs whose *type* attribute is of one of the types specified by sel_p^n).
 - b. If there is a subset of events in $sel_p^n(EH)$ that may be selected by sel_p^n such that $pred_p^n$ is true, add a vertex for the inferred event's EID E_p . In addition, add edges from all events in $sel_p^n(EH)$ to the event E_p .
 - c. For E_p , fill in the quantities for the conditional probabilities according to Eqs. 4, 5, and 6.
4. Calculate the required occurrence probabilities in the probability space defined by the constructed Bayesian network.

The above algorithm describes at a high level the calculation of the required probabilities, omitting the details of several steps. The omitted details include the mechanism for selection of events as indicated by sel_p^n , the evaluation of the predicates defined by $pred_p^n$, and the exact algorithm used to infer the required probabilities from the Bayesian network. In all of these cases, standard algorithms from the domains of deterministic event composition and Bayesian networks may be used and extended. The specific algorithms used for these tasks will determine the complexity of our algorithm. However, the dominant factor will be the calculation of the required probabilities from the Bayesian network, which is known to be computationally expensive.

Handling Uncertainty through Fuzzy Set Theory

Modeling uncertain raw events — An event attribute x is represented (Liu and Jacobsen 2004) in the form “ x is \tilde{A} ,” where \tilde{A} is a fuzzy value with an attached membership function μ . With fuzzy set theory, the event “sensor s temperature reading x is in acceptable range” could be modeled with the following membership function:

$$\mu_{\text{acceptable}}(x) = \begin{cases} 0.15 & \text{if } 37.3 \leq x < 37.4 \\ 0.50 & \text{if } 37.4 \leq x \leq 37.6 \\ 0.15 & \text{if } 37.6 < x \leq 37.7 \\ 0.0 & \text{otherwise} \end{cases}$$

Inference over uncertain events — The membership function of a complex event \hat{e} over n events e_1, \dots, e_n is defined as follows: $M(e_1, \dots, e_n) = R(\mu_A(e_1), \dots, \mu_A(e_n))$, where the relation R determines the method according to which membership values of different fuzzy sets can be combined. Consider once more the event “temperature of sensor s is acceptable” and an additional event “day d is cold” with the following membership function μ_{cold} :

$$\mu_{\text{cold}}(x) = \begin{cases} 0.0 & \text{if } x < 10.0 \\ 0.1 & \text{if } 10.0 \leq x < 15.0 \\ 0.2 & \text{if } 15.0 \leq x < 20.0 \\ 0.7 & \text{otherwise} \end{cases}$$

The complex event “temperature of sensor s is acceptable and day d is cold” can be defined through the following membership function $M(e_1, e_2) = \min(\mu_{\text{cold}}(e_1), \mu_{\text{acceptable}}(e_2))$.

The min operator is the most well-known representative of a large family of operators called t-norms (i.e., triangular norms), routinely deployed as interpretations of fuzzy conjunctions (Hajek 1998; Klir and Yuan 1995). A t-norm $T : [0, 1]^2 \rightarrow [0, 1]$ is a binary operator satisfying the following axioms $\forall x, y, z \in [0, 1]$:

Boundary condition: $T(x, 1) = x$

Monotonicity: $x \leq y \implies T(x, y) \leq T(y, z)$

Commutativity: $T(x, y) = T(y, x)$

Associativity: $T(x, T(y, z)) = T(x, T(y, z))$

A fuzzy aggregate (Klir and Yuan 1995) is an operator $H : [0, 1]^n \rightarrow [0, 1]$ satisfies the following axioms $\forall x_1, \dots, x_n, y_1, \dots, y_n \in [0, 1]$ such that $\forall i \in [1, n], x_i \leq y_i$:

Idempotency: $H(x_1, \dots, x_1) = x_1$

Increasing monotonicity: $H(x_1, \dots, x_n) \leq H(y_1, \dots, y_n)$

H is continuous

The average operator $Ha(\bar{x}) = \frac{1}{n} \sum_1^n x_i$ is an example of fuzzy aggregate operator.

The space of possible computation method is large and additional research is needed to identify the best fuzzy operator for a given complex event.

U

Research Directions

Uncertainty in event streams involves three classes of challenges, namely, modeling, usability, and implementation.

Considering modeling, a single model does not fit all cases. The challenge is to define a flexible generalized model that can accommodate the model best suited for a specific implementation.

A major difficulty with respect to usability is to obtain required patterns and probabilities. As for many other scenarios, specifying correct cases and their associated probabilities is a daunting task even for domain experts. Patterns and probabilities may be automatically generated through machine learning techniques (Turchin et al. 2009); however state-of-the-art in this area supports only mining of simple patterns.

In the implementation area, stream processing engines have reached high levels of scalability and performances (Heinze et al. 2014), a trend that CEP engines must abide to. Therefore, there is a need to develop algorithmic performance improvements (Wasserkrug et al. 2012b) to the current models such as Bayesian networks, which are known to be computationally intensive. Possible additional directions for future work include performance improvements of existing derivation algorithms, either by general algorithmic improvements or by developing domain- and application-specific efficient algorithms.

References

- Arroyo-Figueroa G, Sucar LE (1999) A temporal bayesian network for diagnosis and prediction. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence (UAI'99). Morgan Kaufmann Publishers Inc., pp 13–20
- Bacchus F (1990) Representing and reasoning with probabilistic knowledge: a logical approach to probabilities. MIT Press, Cambridge
- Cugola G, Margara A, Matteucci M, Tamburrelli G (2015) Introducing uncertainty in complex event processing: model, implementation, and validation. Computing 97(2):103–144
- Drakopoulos J (1994) Probabilities, possibilities, and fuzzy sets. *Fuzzy Set Syst* 75:1–15
- Dubois D, Prade H (1988) Possibility theory: an approach to computerized processing of uncertainty. Plenum press, New York
- Etzion O, Niblett P (2010) Event processing in action. Manning Publications Co. Shelter Island, New York, United States
- Gal A, Anaby-Tavor A, Trombetta A, Montesi D (2005) A framework for modeling and evaluating automatic semantic reconciliation. *VLDB J* 14(1):50–67
- Green TJ, Tannen V (2006) Models for incomplete and probabilistic information. Springer, Berlin/Heidelberg, pp 278–296
- Hajek P (1998) Metamathematics of Fuzzy logic. Kluwer Academic Publishers, Dordrecht
- Halpern JY (1990) An analysis of first-order logics of probability. *Artif Intell* 46(3):311–350
- Halpern JY (2003) Reasoning about uncertainty. MIT Press, Cambridge
- Heinze T, Aniello L, Querzoni L, Jerzak Z (2014) Cloud-based data stream processing. In: Proceedings of the 8th ACM international conference on distributed event-based systems, DEBS
- Kanazawa K (1991) A logic and time nets for probabilistic inference. In: Proceedings of the ninth national conference on artificial intelligence (AAAI'91), vol 1. AAAI Press, pp 360–365
- Klir GJ, Yuan B (1995) Fuzzy sets and fuzzy logic: theory and applications. Prentice Hall, Upper Saddle River
- Liu H, Jacobsen HA (2004) Modeling uncertainties in publish/subscribe systems. In: Proceedings of 20th international conference on data engineering, pp 510–521
- Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco
- Raiffa H (1997) Decision analysis: introductory lectures on choices under uncertainty. McGraw-Hill College, New York
- Ré C, Letchner J, Balazinska M, Suciu D (2008) Event queries on correlated probabilistic streams. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD'08). ACM, pp 715–728
- Turchin Y, Gal A, Wasserkrug S (2009) Tuning complex event processing rules using the prediction-correction paradigm. In: Proceedings of the third ACM international conference on distributed event-based systems (DEBS'09), New York. ACM, pp 10:1–10:12
- Wasserkrug S, Gal A, Etzion O, Turchin Y (2008) Complex event processing over uncertain data. In: Proceedings of the second international conference on distributed event-based systems (DEBS'08). ACM, pp 253–264
- Wasserkrug S, Gal A, Etzion O (2012a) A model for reasoning with uncertain rules in event composition systems. CoRR, abs/1207.1427
- Wasserkrug S, Gal A, Etzion O, Turchin Y (2012b) Efficient processing of uncertain events in rule-based systems. *IEEE Trans Knowl Data Eng* 24(1): 45–58
- Widom J, Ceri S (eds) (1994) Active database systems: triggers and rules for advanced database processing. Morgan Kaufmann Publishers Inc. San Francisco, California
- Zadeh L (1965) Fuzzy sets. *Inf Control* 8(3):338–353
- Zhang H, Diao Y, Immerman N (2013) Recognizing patterns in streams with imprecise timestamps. *Inf Syst* 38(8):1187–1211

User Mobility Planning

- ▶ Using Big Spatial Data for Planning User Mobility

Using Big Spatial Data for Planning User Mobility

Mohammad Saiedur Rahaman,
Margaret Hamilton, and Flora D. Salim
School of Science (Computer Science and IT),
RMIT University, Melbourne, VIC, Australia

Synonyms

Context-aware user mobility; Trip context inference; User mobility planning

Definitions

User mobility is the movement of individuals from one place to another. *Trip* is a segment of user mobility. A *context* is any information that can be used to characterize the situation of an entity (i.e., any user), according to Dey and Abowd (2000). *Trip contexts* provide trip-related information to any person planning to make a trip. Further, *route planner* is a system designed to help user mobility which consists of a number of trips. Usually, the route planners present some predefined contexts to a user and provide a route plan between two locations on the basis of user-selected contexts. The *big spatial data* can facilitate diverse sets of trip contexts with their respective geographic location information on the earth's surface which can aid the effective planning of user mobility.

Overview

The study of user mobility is to deal with understanding, analyzing, and modeling the movement

of individuals in the spatial and temporal domains. A meaningful movement (i.e., movement between two meaningful places) of a user is considered user mobility. For example, home and office are two meaningful places. Therefore, the “home to office” and “office to home” movements of a user are called user mobility. In fact, these two types of mobility are very common in our day-to-day life. User mobility can be divided into small segments called “trips.” Essentially, a set of trips connecting two arbitrary places is required to define the user mobility between two meaningful places. Each trip is accomplished using a mode of transport such as bike, car, public transport, or walking. An example of user mobility is illustrated in Fig. 1. We can see that the user mobility between two places p_1 and p_5 consists of a sequence of interconnected trips using different modes of transport. A user can choose his/her preferred mode of transport from the list of available modes. This results in different routes being selected for traveling between p_1 and p_5 . Figure 1 shows three different routes from p_1 to p_5 marked with three different colors (i.e., black, brick red, and blue). Each of these routes has a different combination of transport modes.

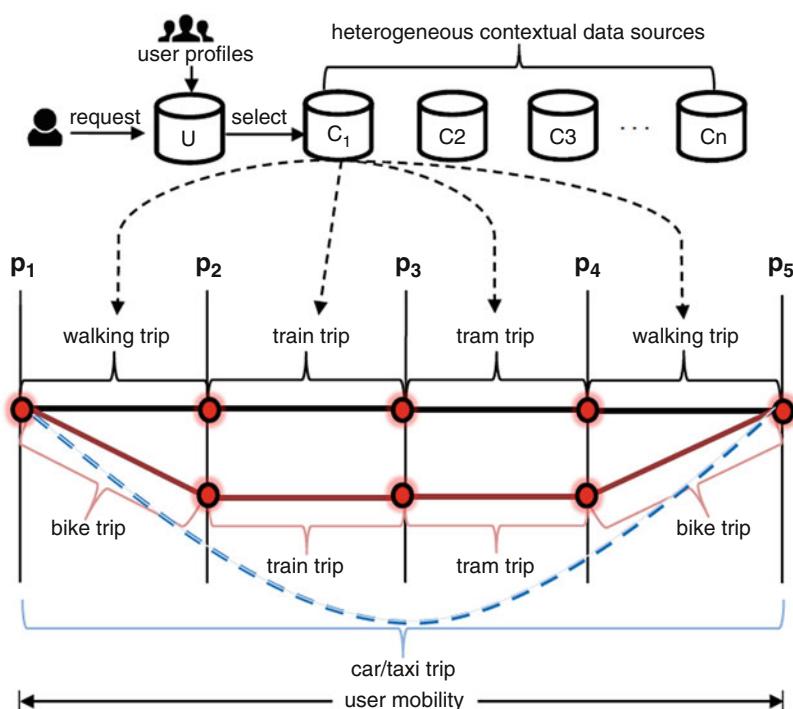
The first two routes (i.e., black and brick red) are for public transport users, and the corresponding sequence of transport modes from p_1 to p_5 is (walk, train, tram, walk) and (bike, train, tram, bike), respectively. Note that these two routes have some common parts in terms of the preferred mode of transport, as seen between p_2 and p_4 . The third route contains only car or taxicab as the preferred mode of transport to travel between p_1 and p_5 .

Route planner systems allow users to plan their mobility between two places. Usually, a route planner integrates trips on the basis of certain predefined user preferences chosen from a list of options and includes the shortest time, shortest distance, minimum number of transport mode changes, and the preferred mode of transport. However, many other situations, if considered, can significantly improve the mobility planning. For instance, an effective mobility plan should include traveler-specific situational

factors (i.e., trip contexts) for each trip selected for integration into the trip plan. These trip contexts can be related to the trip origin location, trip destination location and the route itself. Sometimes, external contexts such as weather can have a notable influence on the trip selection for planning user mobility.

Let us consider two different users and their respective trip contexts. The first user is a person with a wheel chair, and the second is a taxicab driver. While planning a mobility along an arbitrary mobility segment (i.e., trip), the person with a wheel chair needs to consider contexts such as steepness, weather, and road crossing. This user may also consider information about ramps and lifts at the trip end location to ensure comfortable mobility. In contrast, a taxicab driver would consider traffic congestion while determining the preferred mobility segment. The spatial and temporal changes in lucrativeness of mobility segments can also be considered since some mobility segments are more lucrative for a passenger pickup job during weekdays,

while others are so on weekends. The weather can have a direct effect on the state of congestion along a mobility segment. This driver may also prefer a trip with a high likelihood of getting a passenger pickup job at the trip end location. The preference may also include the waiting time before a passenger pickup job with the job likelihood. Therefore, these contexts are very important as they have a considerable effect on these users' choice of trips. Therefore, mobility planning is a challenging issue, as the contexts are diverse among users and user-specific contexts must be taken into consideration for effective planning. An example of context awareness in mobility planning is illustrated in Fig. 1. Here, different user profiles are stored in the user profile database (U). The heterogeneous contexts are stored in separate databases (C_1, C_2, \dots, C_n). Based on a user's request, the appropriate user profile is selected which includes the user-specific context(s). These user-specific contexts are then used to retrieve appropriate contextual information from heterogeneous contextual databases. The



Using Big Spatial Data for Planning User Mobility, Fig. 1 Example of trip contexts in user mobility planning

retrieved information is used for user mobility planning.

For providing effective mobility plans, all of these trip contexts need to be collected, stored, and analyzed. The collection of these context data was not this easy even a decade ago. However, the current era of *big spatial data* enables us to collect and analyze vast amounts of relevant information about many natural or constructed trip contexts with respect to space and time.

Key Research Findings

The big spatial data can provide many important contextual information for user mobility planning and decision making. The availability of various types of contextual information and given the fact that these contexts originate from heterogeneous spatial data sources, various research problems have emerged. Some key research issues include contextual data collection, contextual data fusion and analysis, and context-aware planning using this vast amount of spatial data and taking into account the context information.

Contextual Data Collection

It is important and challenging to collect various contextual data to address diverse sets of user needs for effective mobility planning. The collection of contextual big spatial data is leveraged by the ubiquitousness of sensing devices (i.e., GPS, ticketing systems, and passenger and pedestrian counting systems). This large volume of data related to trip contexts can be used for traveler-specific mobility planning. In recent years, user mobility have been studied using large publicly available datasets such as the Geolife trajectory dataset (Zheng et al. 2008) and the NYC taxi trip dataset (TLC 2016). These datasets include traces of pedestrians, public transport, and taxi trips with timestamps (Zhao et al. 2015). Many research projects have been conducted using these publicly available datasets to develop real-world mobility solutions.

The Geolife GPS trajectory dataset (Zheng et al. 2008) was built using the GPS traces of 182 users between April 2007 and August 2012. A

GPS trace is represented by a sequence of timestamped latitude/longitude points. This dataset contains a total of 17,621 user traces covering a total distance of approximately 1200,000 km. These user traces were recorded at a variety of sampling rates. Approximately 91% of the traces were recorded every 15 s, which was equivalent to 510 m per spatial point. The GPS traces recorded in this dataset represent different types of outdoor movements of users, such as “go to work,” “go home,” “shopping,” “dining,” “sightseeing,” “bicycling,” and “hiking.” This trajectory dataset is used in many research fields ranging from user mobility pattern mining, activity recognition and mobility planning to urban planning, and location recommendation.

The NYC taxi trip dataset (TLC 2016) is a real-world taxi trip dataset built by the New York City’s Taxi & Limousine Commission (TLC). In New York City, 0.5 million taxi trips are generated on an average per day, totaling 175 million trips per year by 13,000 operating taxis. Each trip record in this dataset describes a trip by its timestamped start and end locations, the trip distance, total number of passengers on board during the trip, fare type, fare and tip amount, and the taxi’s unique identification number called medallion.

Contextual Data Fusion and Analysis

A fusion of trip contexts is required after the collection of the context information from various sources. This fusion is required to better represent the effects of other trip contexts for planning the user mobility between two places. Thus far, a number of data fusion methodologies have been developed for cross-domain data fusion (Zheng 2015). Further, a context representation framework can be used to deal with this issue (Rahaman et al. 2018). Context representation techniques for trip planning can leverage the existing data modeling techniques.

Sometimes, predictive analytics is used for an inference-based representation of trip contexts for user mobility. The predictive analytics provides a detailed analysis and prediction of future trip contexts. To provide efficient prediction outcomes, different machine learning techniques are used. These machine learning

techniques are trained with historical data to gain appropriate knowledge for a prediction task. However, the predictive analytics need to deal with various issues such as data sparsity and imbalance problems. Some of the trip contexts can be imbalanced (i.e., very infrequently occurring contexts as compared to the others) which makes the predictive analytics more complex. Many researchers have addressed this issue. Various sampling techniques can be used to deal with this issue (He and Garcia 2009). Some researchers have also investigated how expert knowledge can be collected and incorporated into the problem domain. A new strategy to incorporate expert knowledge for enhanced trip context prediction was proposed by Rahaman et al. (2017b).

Context-Aware Planning

For providing effective mobility plans, user-specific trip contexts need to be considered in such a way that the wide range of user-specific trip contexts can be addressed. However, this is a challenging issue to deal with. Thus far, several algorithmic approaches have been proposed to handle user-defined contextual preferences while providing plans for user mobility. Most of these approaches provide mobility plans considering only the context selected by the user. Some route planner systems consider more than one trip context at a time while providing mobility plans. Some researchers also consider user-specific trip-special contexts for recommending mobility plans.

Examples of Application

Big spatial user mobility data have been used in many real-world applications. The applications related to user mobility planning can be categorized as follows: trip context inference, contextual predictive analytics, and contextual modeling.

- In context inference, various trip contexts are retrieved from the different big spatial

datasets, which may include the fusion of various heterogeneous datasets.

- The predictive analytics of trip contexts provides predictions about future trip contexts by using different statistical, machine learning, and data mining techniques to analyze the current and historical trip events from big spatial user mobility datasets.
- In contextual modeling, the trip context(s) are taken into consideration to provide mobility plans for different users. Usually, the context information is represented in a graph or matrix form before applying the route planning techniques.

In many cities such as Singapore and New York, taxicabs are regarded as the preferred mode of transport for user mobility. Here, big spatial data have been utilized to predict user-specific taxi trip contexts including taxi drivers and passengers. For instance, it is important for a taxi driver to predict the queue context information at the trip end location before making a trip. A queue context prediction framework was proposed by Rahaman et al. (2017a) which leveraged the big spatial data generated by the taxi trip logs in New York City. Some other research projects aimed at providing information about trip contexts. For instance, techniques for finding the lucrativeness of passenger pickup locations were developed by Hwang et al. (2015) and Yuan et al. (2011). By analyzing the historical mobility data of taxicabs, techniques to find profitable routes for cruising before passenger pickup were proposed by Ge et al. (2010) and Dong et al. (2014). By considering the passengers' trip contexts, an on-demand trip context information system was developed by Balan et al. (2011) whose aim was to provide trip contexts such as approximated fare and trip time prior to the trip. Another research proposed a model for passenger wait time prediction before a taxi ride (Qi et al. 2013) which utilized a large number of historical taxi GPS traces.

Contextual modeling is used for providing effective mobility planning and recommendations to the users. Many studies were conducted to rec-

Using Big Spatial Data for Planning User Mobility, Table 1 Contexts used in different applications and relevant publicly available user mobility datasets

Datasets	Context	Application
Geolife GPS trajectory ^a	Congestion, popularity of location,	Traffic prediction, route recommendation, location recommendation
NYC taxi trip log ^b	Queue context, wait time	Queue context and wait time prediction
Foursquare ^c	Crime, city dynamics	Crime Prediction, location recommendation, route recommendation
Bike share ^d	Demand	Demand prediction

^a<https://microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/>

^bhttp://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

^c<https://developer.foursquare.com/>

^d<http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>

ommend routes considering two of the most common user preferences: distance and time. Many other research works focused on route planning and recommendation (Sasaki and Takama 2013; Völkel and Weber 2008; Rahaman et al. 2017c) by considering other user-specific contexts leveraging the big spatial data sources. A route recommendation system was developed by Sasaki and Takama (2013) which considered three criteria to recommend a route, namely, safety, amenity, and walkability of the route. A client/server system to annotate multimodal geographical data and to provide mobility service to mobility impaired pedestrians was proposed by Völkel and Weber (2008). Multi-criteria route planning for people with special needs was proposed by Rahaman et al. (2017b). Route recommendation considering the scenic beauty of the route was proposed by Quercia et al. (2014), while relative safety was considered by Elsmore et al. (2014) for mobility planning. Table 1 summarizes the use of contexts in different applications using public user mobility datasets.

formation is collected through direct crowd participation such as safety or scenic beauty ratings of a mobility segment. This information is a good data source for making context-aware mobility plans, as the information contains actual user perceptions. However, it is challenging to ensure the data quality of such crowd-generated context information. Future research can address this issue.

Another research direction is related to the diversity of spatial contextual data sources. These contextual data are heterogeneous, i.e., sourced from different domains and locations. These data have different sampling rates and representations. Advanced techniques for heterogeneous data fusion are required for the effective user mobility planning.

Many trip contexts require efficient predictive analytics with big spatial data for user mobility. However, predictive analytics techniques are usually domain specific. Therefore, appropriate domain adaptation is required for the analytics techniques adopted from other domains.

Future Directions for Research

There are several directions for future research on user mobility planning using big spatial. The three most important aspects are data quality, data fusion, and effective predictive analytics.

In the era of big data, the spatial contextual information is mainly collected through autonomous loggers. However, some contextual in-

Cross-References

- ▶ [Big Data Analysis for Smart City Applications](#)
- ▶ [Big Data in Smart Cities](#)
- ▶ [Data Fusion](#)
- ▶ [Spatiotemporal Data: Trajectories](#)

Acknowledgements This work was supported by the Australian Research Council's Linkage Projects funding scheme (project number: LP120200305).

References

- Balan RK, Khoa NX, Jiang L (2011) Real-time trip information service for a large taxi fleet. In: Proceedings of the international conference on mobile system, applications, and services (MobiSys), Bethesda
- Dey A, Abowd G (2000) Towards a better understanding of context and context-awareness. In: CHI 2000 workshop on the what, who, where, when, and how of context awareness
- Dong H, Zhang X, Dong Y, Chen C, Rao F (2014) Recommend a profitable cruising route for taxi drivers. In: Proceedings of the 17th international conference on intelligent transportation systems (ITSC), Qingdao, pp 2003–2008
- Elsmore S, Sebastian I, Salim F, Hamilton M (2014) Vdim: vector-based diffusion and interpolation matrix for computing region-based crowdsourced ratings: towards safe route selection for human navigation. In: Proceedings of the 13th international conference on mobile and ubiquitous multimedia, pp 212–215
- Ge Y, Xiong H, Tuzhilin A, Xiao K, Gruteser M, Pazzani MJ (2010) An energy-efficient mobile recommender system. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining (KDD), Washington DC, pp 899–907. <https://doi.org/10.1145/1835804.1835918>
- He H, Garcia EA (2009) Learning from imbalanced data. IEEE Trans Knowl Data Eng 21(9):1263–1284
- Hwang RH, Hsueh YL, Chen YT (2015) An effective taxi recommender system based on a spatio-temporal factor analysis model. Inf Sci 314:28–40. <https://doi.org/10.1016/j.ins.2015.03.068>
- Qi G, Pan G, Li S, Wu Z, Zhang D, Sun L, Yang LT (2013) How long a passenger waits for a vacant taxi – large-scale taxi trace mining for smart cities. In: Proceedings of the IEEE international conference on green computing and communications (GCC), Beijing, pp 1029–1036
- Quercia D, Schifanella R, Aiello L (2014) The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In: Proceedings of the 25th ACM conference on hypertext and social media (HT'14), pp 116–125
- Rahaman MS, Hamilton M, Salim FD (2017a) Predicting imbalanced taxi and passenger queue contexts in airport. In: Proceedings of the Pacific Asia conference on information systems (PACIS)
- Rahaman MS, Hamilton M, Salim FD (2017b) Queue context prediction using taxi driver knowledge. In: Proceedings of the knowledge capture conference (K-CAP'17)
- Rahaman MS, Mei Y, Hamilton M, Salim FD (2017c) Capra: a contour-based accessible path routing algorithm. Inf Sci 385–386:157–173
- Rahaman MS, Hamilton M, Salim FD (2018) Coact: a framework for context-aware trip planning using active transport. In: Proceedings of the percom workshops
- Sasaki W, Takama Y (2013) Walking route recommender system considering saw criteria. In: 2013 conference on technologies and applications of artificial intelligence, pp 246–251
- TLC (2016) NYC taxi & limousine commission: NYC taxi trip data (online). http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. Last visited on 01 Feb 2016
- Völkel T, Weber G (2008) Routecheckr: personalized multicriteria routing for mobility impaired pedestrians. In: Proceedings of the 10th international ACM SIGACCESS conference on computers and accessibility, pp 185–192
- Yuan J, Zheng Y, Zhang L, Xie X, Sun G (2011) Where to find my next passenger? In: Proceedings of the 13th international conference on ubiquitous computing (UbiComp), Beijing, pp 109–118
- Zhao K, Musolesi M, Hui P, Rao W, Tarkoma S (2015) Explaining the power-law distribution of human mobility through transportation modality decomposition. Nature scientific reports 5
- Zheng Y (2015) Methodologies for cross-domain data fusion: an overview. IEEE Trans Big Data 1(1):16–34
- Zheng Y, Li Q, Chen Y, Xie X, Ma WY (2008) Understanding mobility based on gps data. In: Proceedings of ACM conference on ubiquitous computing (UbiComp'08), pp 312–321

V

Validation

► [Auditing](#)

Virtual Distributed File System: Alluxio

Calvin Jia and Haoyuan Li
Alluxio Inc., San Mateo, CA, USA

Synonyms

[Tachyon](#)

Definitions

Alluxio (2018) is the world’s first memory speed virtual distributed storage. Alluxio is an open source project with a community of over 700 contributors from over 150 organizations. In the big data stack, it is a layer between applications such as Apache Spark (2018) and Apache MapReduce (Apache Hadoop 2018a) and data stores such as Amazon S3 (Amazon 2018) (Simple Storage System) and Apache HDFS (Apache Hadoop 2018b) (Hadoop Distributed File System).

Alluxio presents a set of disparate data stores as a single file system, greatly reducing the

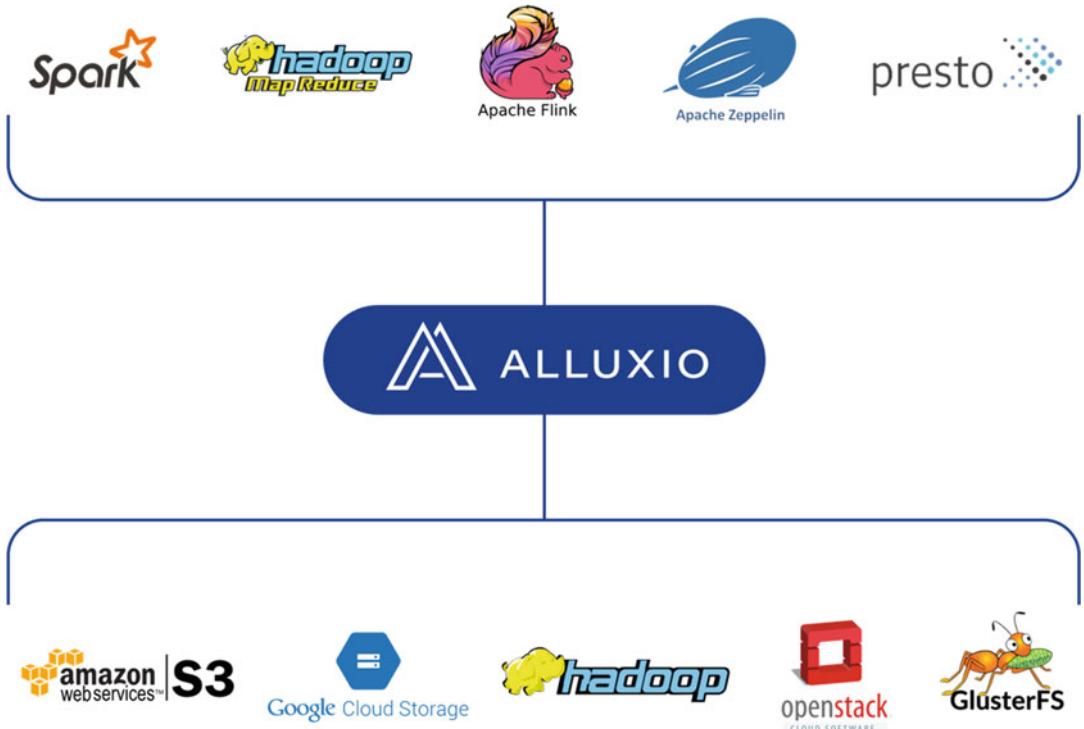
complexity of storage APIs (Application Programming Interface), locations, and semantics exposed to applications. For example, an object storage in the cloud and an on-premise distributed file system both appear to applications as subtrees in the Alluxio file system namespace.

Alluxio is designed with a memory centric architecture, enabling applications to leverage memory speed I/O by simply using Alluxio. Because Alluxio is between compute and storage, it serves as a transparent cache that enables applications to benefit from data sharing and locality.

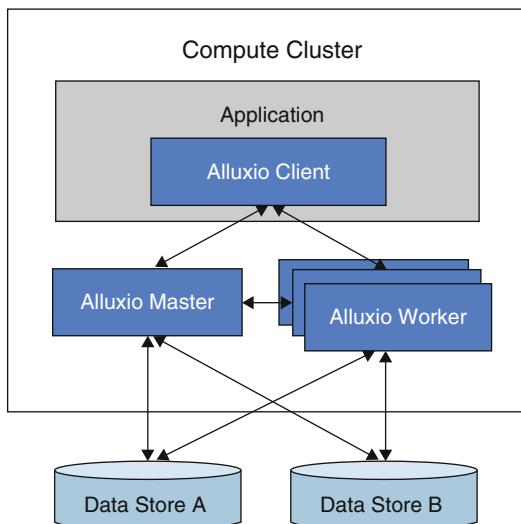
By being a layer between applications and data stores, Alluxio enables rapid innovation in both compute and storage by abstracting basic functionality and compatibility.

Overview

Alluxio is a Java-based distributed system with a master–worker architecture. Master nodes are responsible for handling metadata requests and cluster registration. Workers are responsible for handling data requests and transferring data from Alluxio storage or data stores to clients. Clients are part of applications that use Alluxio. Finally, the under storage is comprised of different data stores which house the persistent copies of the data. Figure 1 above shows the general interaction pattern between Alluxio and other projects in the big data ecosystem, and Fig. 2 below shows the communication and deployment pattern of Alluxio with respect to its internal components.



Virtual Distributed File System: Alluxio, Fig. 1 Alluxio and the ecosystem



Virtual Distributed File System: Alluxio, Fig. 2
Alluxio system components

Clients

The Alluxio client is a library that is used by applications to interface with the servers. Client

applications have several interfaces available to them – an Alluxio Java interface, a Hadoop compatible file system interface, a S3 compatible object store interface, a FUSE (Filesystem in Userspace) interface, and a REST (Representational State Transfer) interface. Alluxio supports many interfaces to give application developers the most flexibility when working with Alluxio.

The Alluxio Java interface is preferred for the best performance and functionality. The newest features will always be available in the Java interface first, and it will have the smallest overhead of all the clients.

Because Alluxio supports a Hadoop-compatible file system interface, existing applications in the ecosystem can leverage Alluxio by providing the client binary and switching input paths to an “alluxio://” scheme. This is desirable, as no code changes are needed for Alluxio to be used with applications that were already using a Hadoop-compatible file system interface.

Similarly, the S3 interface allows applications that previously used S3 as storage to switch to Alluxio with minimal code changes. An object storage interface can be attractive for applications dealing with media data. While Alluxio provides the same interface, it provides characteristics such as strong consistency and memory speed I/O that are not commonly provided by object stores.

The FUSE interface allows applications previously working with local storage to seamlessly migrate to using Alluxio. This is even more transparent than the Hadoop integration, and no binary or path changes are required. A separate Alluxio FUSE process is responsible for redirecting requests to the mounted path to Alluxio. The FUSE interface allows applications previously working with only local data to expand to all the data repositories available in an organization.

Finally, the REST interface is a convenient and ubiquitous interface for applications that may only need to do simple operations with Alluxio. This interface can be used by any application that can send HTTP requests.

Masters

The Alluxio master is a quorum of nodes (possibly only one) that record the state of the Alluxio file system and cluster. Only one master node is actively serving requests at a time, the other nodes participate in keeping the journal up-to-date as well as act as hot failovers if the active master crashes. Quorum membership and leader election are managed through Apache Zookeeper (2018).

The Alluxio master manages worker membership, registering a unique id for each worker. Each Alluxio worker is expected to only associate with one Alluxio master. Alluxio masters also assign unique ids to workers and keep track of the data available on each worker.

Alluxio master records the metadata for data blocks which make up files. Blocks are the smallest unit of data in Alluxio, usually ranging from tens to hundreds of megabytes. Each block has a unique id but can be stored on zero, one, or many worker nodes.

One or more blocks comprise the data portion of a file. The master also keeps track of the file system tree information required to map a human readable path to a specific file. Along with path information, the Alluxio master also keeps various file level metadata such as permissions and modification time.

The Alluxio master is designed to accept incoming requests but not make any outgoing requests, aside from responses. This allows clients and workers to freely join or leave the cluster. Clients typically communicate with the master to make logical updates to the metadata tree or create new files. These requests are independent and stateless, enabling the master to scale up to thousands of concurrent clients. Workers have a periodic heartbeat with the master for as long as they are part of the cluster. The heartbeat is low cost, and Alluxio clusters have scaled to a thousand nodes in production environments.

The master is also responsible for keeping a journal of events. This is for fault tolerance guarantees; whenever the master restarts, it recovers the previous metadata state including the file system tree and block list from the journal. The journal is typically stored on distributed storage to avoid a single point of failure, for example, if a disk were lost. Protobufs (Google 2018) are used to serialize journal entries with the added benefit of being easily extended to support future use cases.

Typical deployments have two or more master nodes for fault tolerance. The number of files the Alluxio master supports is proportional to the memory allocated to it. A single master node can scale up to hundreds of millions of files. Similarly, allocating more CPUs to the master allows for lower response latency in highly concurrent environments.

Workers

Worker nodes make up the bulk of an Alluxio cluster. These nodes are responsible for managing local storage (Alluxio storage) and providing data access to the data stores Alluxio abstracts to applications. The Alluxio worker process is lightweight and does not rely on persistent data; the system is designed to tolerate the complete

loss of a worker and its local storage. Because of the worker's small footprint, worker processes run colocated with applications.

Deploying the worker process colocated with applications greatly improves the maximum possible read/write throughput to Alluxio. Alluxio has optimizations to allow clients to read and write at memory speed if the client is running on the same node. In workloads that require a large amount of I/O, reducing the network traffic and serving data locally significantly improve performance.

Alluxio workers can be deployed on dedicated nodes as well. This has the benefit of isolating workers, which is helpful in cases where the application nodes are ephemeral or not committed to the cluster. For example, a single set of Alluxio worker nodes could serve several isolated compute clusters belonging to different users. In this way, the compute clusters do not need to be concerned about having any resources taken by Alluxio due to the workload of other clusters.

The primary function of Alluxio workers is to serve data requests from clients. It can do so in one of two ways, either directly from Alluxio local storage or by fetching data from an underlying data store, for example, Amazon S3. Data fetched from an underlying data store can be cached in Alluxio local storage, so subsequent accesses, even from completely unrelated applications, can be accelerated. This is especially helpful in workloads which focus on a narrow window of data among a larger total dataset, for example, when commonly analyzing the last 30 days of data while keeping several years of data available.

Examples of Application

Enabling Decoupled Compute and Storage

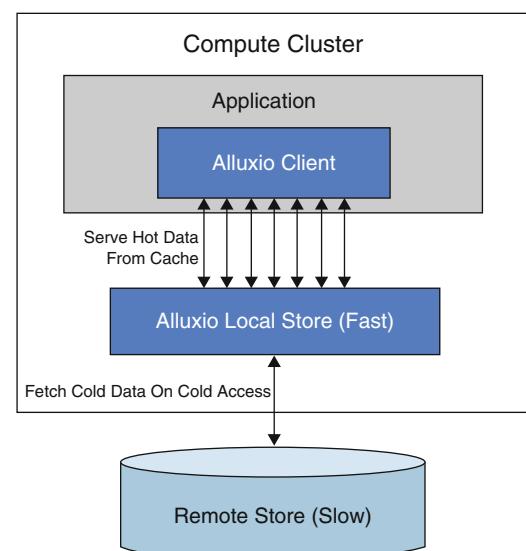
Decoupling compute and storage, and thus allowing optimal allocation of storage and compute resources, brings considerable cost savings as well as ease of innovation. There are two substantial challenges to this architecture, the performance impact due to lower bandwidth and higher latency and the limitations of writing an application against a particular storage interface.

By providing a lightweight cache colocated with the application, Alluxio remedies the performance impact of having remote storage. Organizations from various industry sectors, including some of the largest Internet companies in the world (Alluxio and Baidu 2016), have used Alluxio to great effect in this manner (see Fig. 3 below for a visual representation).

Alluxio also solves the interface limitation problem by allowing applications to choose from a rich set of standard APIs and connect seamlessly to any data store integrated with Alluxio. This greatly improves the flexibility of application and storage teams. Many companies, including the largest geospatial data distributor (ESRI 2017), use Alluxio to logically decouple compute and storage to enable rapid innovation in the storage and application layers.

Accelerating Data Analytics

Faster time to insight for organizations translates to better-informed decisions, which provides a crucial competitive advantage. The Alluxio memory centric architecture enables top travel booking companies (Alluxio and Qunar 2017) to quickly and accurately analyze user



Virtual Distributed File System: Alluxio, Fig. 3
Alluxio accelerates data access by caching hot data

behavior. This analysis then leads to better recommendations and user targeting.

In addition to accelerating I/O, Alluxio enables previously isolated computation to benefit from the data access patterns of other applications. The ability to share data between applications has led to even further performance gains and less data movement. Figure 3 below shows this for a single application, but multiple applications can utilize the same Alluxio local store.

Data Access for Machine Learning

Machine learning, especially deep learning, is reliant on the available training datasets. Often, it is not the algorithm but the training and validation data that produces the best models. Machine learning workloads have mostly been written with local data in mind, whereas the data necessary to fuel the models are stored in scattered data repositories. By using Alluxio, AI-focused companies (Liu and Dawei 2017) can easily expand the datasets available for training and validation to all the data repositories owned by the organization (see Fig. 4 below for a visual representation). This is a big win over traditional approaches which required complex extraction,

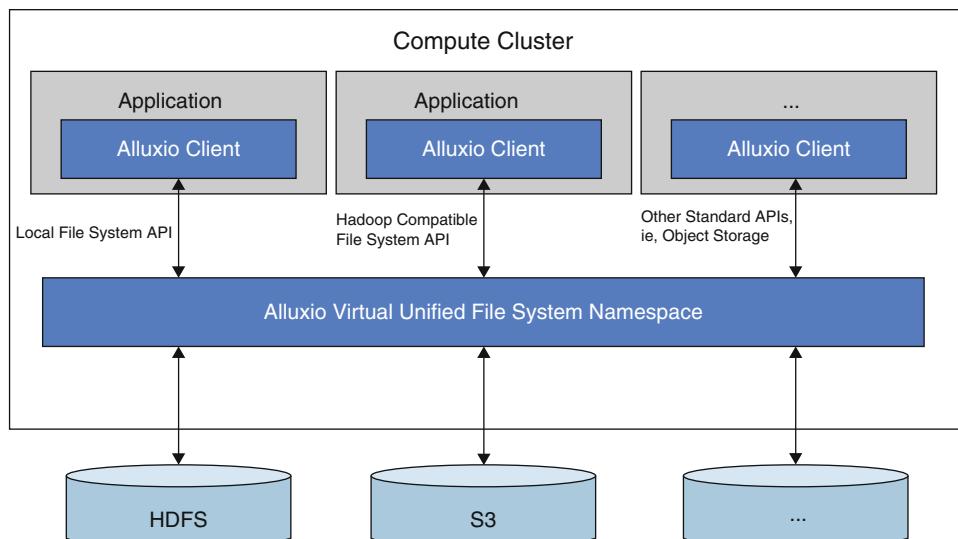
transform, load (ETL) and data tracking to ensure the cleanliness and availability of datasets.

Replacing ETL with Unification

ETL jobs are costly and hard to maintain. Data must be copied from a master source, and any updates must be propagated to the copies to ensure data integrity. Each application or group using the data must also have a separate version of the data, leading to more storage and maintenance requirements.

Alluxio provides applications with virtualization and unification. Organizations such as top financial companies (Alluxio and Barclays 2016) benefit from the huge improvement in manageability and performance gained by using Alluxio as opposed to ETL processes.

Virtualization allows ETL to be transparent, both in terms of any effort required to obtain the data as well as removing the need for users to manage copies of the data. With Alluxio, there is always only one master copy that Alluxio may end up caching at various points. These cached versions are automatically synchronized with the underlying storage in the case of updates or other changes when a client accesses them.



Virtual Distributed File System: Alluxio, Fig. 4 Alluxio provides standard APIs to applications and abstracts disparate data stores in a single unified namespace

Alluxio also provides unification, enabling applications to access various datasets across different storages, without needing to understand where the master data is located. This is especially helpful as organizations try to derive more insights from different data repositories.

Future Research Directions

As more applications and data stores use Alluxio, more sophisticated interfaces will be desired to optimize performance and ease of use. For example, Alluxio currently provides a rich set of file-based APIs to access data, but a large number of applications would prefer APIs such as key-value or SQL.

Alluxio currently acts as a data access layer, providing efficient and simple access to data regardless of data store. As the single layer for data access, Alluxio is in a unique position to collect data metrics and usage patterns. These can give users insight into their data and can also be used by Alluxio to do optimizations in the background. These optimizations could range from data prefetching to data modifications like compression. Many of these optimizations could also be pushed down into the data stores if the data stores support them natively.

Cross-References

- ▶ [Apache Spark](#)
- ▶ [Caching for SQL-on-Hadoop](#)
- ▶ [Distributed File Systems](#)
- ▶ [Hadoop](#)

References

- Alluxio (2018) Alluxio. <https://alluxio.org>
 Alluxio, Baidu (2016) Baidu queries data 30 times faster with Alluxio. <http://alluxio-com-site-prod.s3.amazonaws.com/resource/media/Baidu-Case-Study.pdf>
 Alluxio, Barclays (2016) Making the impossible possible w/ Alluxio. http://alluxio-com-site-prod.s3.amazonaws.com/resource/media/Making_the_Impossible_Possible_w_Alluxio.pdf

- Alluxio, Qunar (2017) Qunar performs real-time data analytics up to 300x faster with Alluxio. http://alluxio-com-site-prod.s3.amazonaws.com/resource/media/Qunar_Performs_Real-Time_Data_Analytics_up_to_300x_Faster_with_Alluxio.pdf
 Amazon (2018) Amazon S3. <https://aws.amazon.com/s3>
 Apache Hadoop (2018a) Apache Hadoop MapReduce. <http://hadoop.apache.org>
 Apache Hadoop (2018b) Apache Hadoop distributed file system. <http://hadoop.apache.org>
 Apache Spark (2018) Apache Spark. <https://spark.apache.org>
 Apache Zookeeper (2018) Apache Zookeeper. <https://zookeeper.apache.org>
 ESRI (2017) ArcGIS and Alluxio. <https://alluxio.com/resources/arcgis-and-alluxio-using-alluxio-to-enhance-arcgis-data-capability-and-get-faster-insights-from-all-your-data>
 Google (2018) Protocol buffers. <https://developers.google.com/protocol-buffers>
 Liu S, Dawei S (2017) PerceptIn robotics get a performance boost from Alluxio distributed storage. <http://thenewstack.io/powering-robotics-clouds-alluxio>

Virtualized Big Data Benchmarks

Tariq Magdon-Ismail
 VMware Inc., Palo Alto, CA, USA

Synonyms

[Cloud big data benchmarks](#); [Virtualized hadoop benchmarks](#)

Definitions

Virtualized big data benchmarks measure the performance of big data processing systems, such as Hadoop, Spark, and Hive, on virtual infrastructures (Ivanov et al. (2014)) (The term *virtual infrastructure* could either mean on-premise or cloud infrastructure. While virtualization is not strictly necessary to support cloud computing, it is typically a foundational element of all major

cloud computing services.). They are important for making quantitative and qualitative comparisons of different systems.

Historical Background

Big data applications are resource intensive and, as such, have historically been deployed exclusively on dedicated physical hardware clusters, i.e., bare-metal systems. As big data processing moved out of the specialized domain of Web 2.0 companies into the mainstream of business-critical enterprise applications, enterprises have been looking to take advantage of the numerous benefits of virtualization such as elasticity (Magdon-Ismail et al. (2013)), scalability, agility, multi-tenancy, security, ease of maintenance, and cost savings due to consolidation and better resource utilization. As the prevalence of virtualized big data workloads increases, so does the importance of virtualized big data benchmarks.

Foundation

In the early 2000s, early adopters advised against virtualizing big data workloads, citing I/O performance degradation since big data systems were originally implemented to run in a bare-metal environment. This is similar to the situation that prevailed in the early 1990s, when data centers relied almost entirely on physical servers for traditional applications. That scenario changed completely with the entry of server virtualization and hypervisors in the early 2000s, and today over 75% of all data centers run virtualized server environments.

Virtualization introduces a degree of separation between the compute and storage layers that is contrary to the traditional method of tightly coupling and colocating a compute task with the data it needs, using bare-metal physical servers and direct-attached storage (DAS). While historically this was a requirement for performance given the prevalence of 1 G Ethernet, innovations in network bandwidth have caused speeds to double and quadruple over the recent years, while

disk bandwidth has not progressed at the same rate. Moreover, as data centers rapidly upgrade from 1 G Ethernet switches to 10 G and even 40 G or 100 G switches, network bandwidth will no longer be a bottleneck for virtualized I/O performance.

With advances in hardware and virtualization technology, the overhead of virtualizing big data workloads has been significantly reduced for most use cases, and, in fact, performance could be even better than a physical cluster because of the better resource utilization achieved with virtualization (Buell 2013). However, the performance impact of virtualization varies depending on the workload, and virtualized big data benchmarks play an important role in evaluating and understanding the performance characteristics of these resource-intensive workloads on a virtualized infrastructure.

There could be subtle changes to the durability and availability guarantees provided by the big data processing software platform when run on a virtual infrastructure (compared to bare metal) depending on how the components of the platform are deployed (HADOOP-8468 (2018)). While the benchmark code might not need to be modified to run on virtualized systems, if the underlying physical infrastructure is not carefully considered, the benchmarked system might not have the equivalent durability/availability characteristics compared to that of a physical deployment. For example, the same version of the Hadoop TeraSort benchmark (O’Malley 2008) that runs on bare-metal hardware will successfully run unmodified on a virtual infrastructure, but physical topology needs to be carefully considered when deploying the Hadoop DataNode (HDFS 2018) virtual machines (VMs). Deploying Hadoop such that data block replication occurs between Hadoop DataNode VMs residing on the same physical server would result in better performance, but if a single server fails, data will be lost, defeating the purpose of, and the guarantees provided by, Hadoop’s distributed file system (HDFS 2018).

Some industry standard benchmarks such TPCx-HS (2018) and TPCx-BB (2018) have stringent full disclosure and audit/review

requirements to make sure that durability and availability guarantees are maintained on a virtual infrastructure the same way as they are in a physical environment.

Key Applications

Big data applications span a wide gamut of applications, which include:

- Web crawling and/or text processing
- Log and/or clickstream analysis
- Sophisticated data mining and analytics
- Machine learning and AI
- General archiving, including relational/tabular data for purposes such as compliance

Virtualized big data benchmarks are designed to measure the performance of these applications on big data processing systems running on virtual infrastructure. While the modeled application workload varies depending on the application domain and software stack being targeted, the goal of these benchmarks is to facilitate a quantitative and qualitative comparison of different systems.

Cross-References

- [Apache Spark](#)
- [Benchmark Harness](#)
- [Big Data and Exascale Computing](#)
- [Big Data Architectures](#)
- [Big Data in the Cloud](#)
- [Cloud Big Data Benchmarks](#)
- [Computer Architecture for Big Data](#)
- [Hadoop](#)
- [Metrics for Big Data Benchmarks](#)
- [Performance Evaluation of Big Data Analysis](#)
- [Scalable Architectures for Big Data Analysis](#)
- [Storage Technologies for Big Data](#)
- [System Under Test](#)
- [TPC](#)
- [TPCx-HS](#)

References

- Buell J (2013) Virtualized hadoop performance with VMware vSphere 6 on high-performance servers. Tech White Pap. VMware Inc.
- HADOOP-8468 (2018) Umbrella of enhancements to support different failure and locality topologies. <https://issues.apache.org/jira/browse/HADOOP-8468>. Accessed 17 Jan 2018
- HDFS Architecture Guide (2018) https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Accessed 22 Jan 2018
- Ivanov T, Zicari RV, Izberovic S, Tolle K (2014) Performance evaluation of virtualized hadoop clusters. ArXiv Prepr. ArXiv14113811
- Magdon-Ismail T, Nelson M, Cheveresan R, Scales D, King A, Vandrovec P, McDougall R (2013) Toward an elastic elephant enabling hadoop for the cloud. VMware Tech J
- O'Malley O (2008) TeraByte sort on apache hadoop. White Pap. Yahoo! Inc.
- TPCx-BB (2018) Benchmark. <http://www.tpc.org/tpcx-bb/default.asp>. Accessed 17 Jan 2018
- TPCx-HS (2018) Benchmark. <http://www.tpc.org/tpcx-hs/default.asp?version=2>. Accessed 17 Jan 2018

Virtualized Hadoop Benchmarks

- [Virtualized Big Data Benchmarks](#)

Visual Analytics

- [Big Data Visualization Tools](#)

Visual Data Exploration

- [Visualization Techniques](#)

Visual Exploration

- [Big Data Visualization Tools](#)

Visual Graph Querying

Sourav S. Bhowmick¹ and Byron Choi²

¹Nanyang Technological University, Singapore, Singapore

²Hong Kong Baptist University, Hong Kong, China

Definitions

Visual querying of graphs: Formulation of a graph query by drawing it using a visual query interface.

Overview

Querying graphs has emerged as an important research problem due to the prevalence of graph-structured data in many real-world applications (e.g., social networks, road networks, collaboration networks, cheminformatics, bioinformatics, computer vision). At the core of many of these applications lies a common and important query primitive called *subgraph search*, which retrieves one or more matching subgraphs or data graphs containing *exact* (Han et al. 2013; Yan et al. 2004) or *approximate* (Yan et al. 2005; Tian and Patel 2008) match of a user-specified query graph (aka subgraph query). Since the last decade, a number of graph query languages (e.g., SPARQL, Cypher) have been proposed to facilitate textual formulation of subgraph queries. All these languages assume that a user has programming and debugging expertise to formulate queries correctly in these languages. Unfortunately, this assumption makes it harder for nonprogrammers to take advantage of a subgraph search framework as it requires significant time and effort to learn these languages. For example, domain experts such as chemists cannot be expected to learn the complex syntax of a graph query language in order to formulate meaningful queries over a chemical compound database such as *PubChem* (<http://pubchem.ncbi.nlm.nih.gov/>) or *eMolecule* (<https://www.emolecules.com/>). Hence, it is im-

portant to devise easy and intuitive techniques that reduce the burden of query formulation and thus increase the usability of graph databases.

Fortunately, unlike SQL, graph queries are more intuitive to draw than to compose them in textual format. Consequently, visual query interfaces (aka GUI) that can enable an end user to draw a subgraph query interactively have gained increasing attention in recent times from both academia and industry (e.g., *PubChem*, *eMolecule*, *DrugBank*). This has recently paved the way for research in a variety of directions that are driven by visual query interfaces such as novel visual query processing paradigm and interactive guidance to users by providing relevant and timely feedback and suggestions during query formulation, exploration, and visualization of graph query results, among others (Bhowmick et al. 2017a).

This chapter gives an overview to the topic of visual graph querying, discussing the state of the art in the industry and in the academic world. In particular, we emphasize research efforts that aim to bridge two traditionally disparate topics in computer science, namely, graph querying and human-computer interaction (HCI). In the next section, we present an overview of the key problems and solutions in the arena of visual graph querying. Next, in section “An Example” we present an example to illustrate them. The last section concludes this chapter by highlighting future research directions.

Key Research Findings

Our discussion on research findings follows a top-down approach, starting from visual query formulation, proceeding to visual query processing, and concluding with exploration of query results.

Visual Query Formulation

There are mainly three popular approaches for formulating visual subgraph queries. In the *edge-at-a-time* approach (e.g., Jin et al. 2012), a query is incrementally constructed by adding one edge

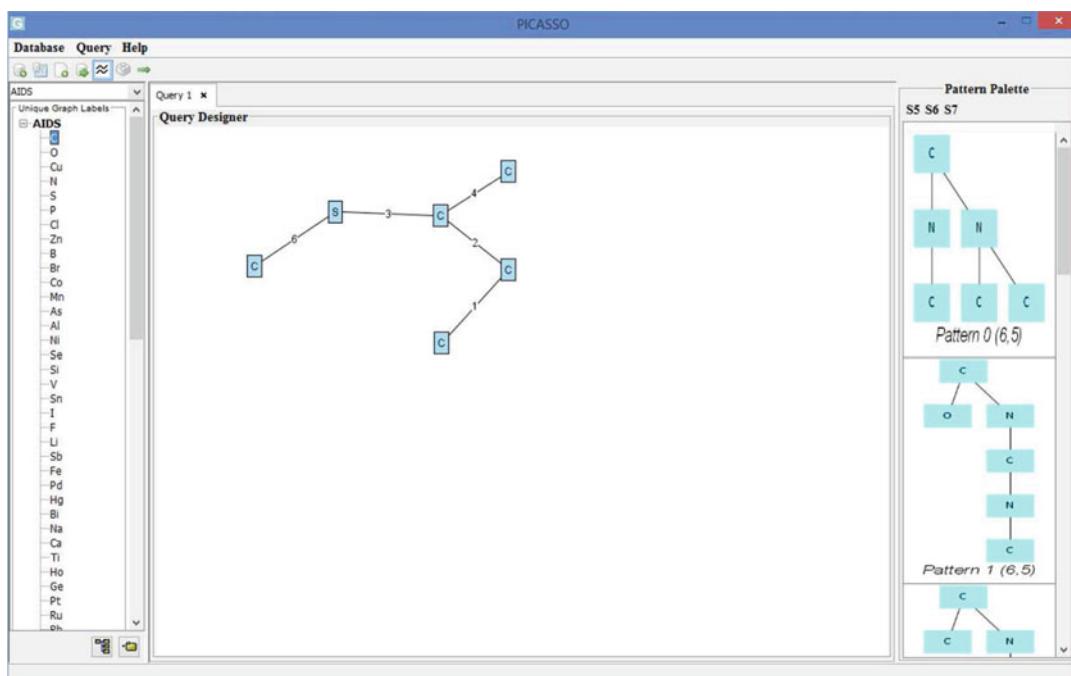
at a time. Note that it may be time-consuming to formulate a query with a large number of edges using this approach. Hence, in the *pattern-at-a-time* approach, one may compose a visual query by dragging and dropping canned patterns or subgraphs (e.g., benzene, chlorobenzene patterns) that are available on the GUI in addition to single edge construction. Figure 1 depicts an example of such an interface. Observe that this approach is more efficient than the former as it typically takes lesser time to construct a query. For instance, instead of drawing six edges incrementally to construct a benzene ring in a query, we can construct it with a single click and drag if it is available as a canned pattern.

Furthermore, the pattern-at-a-time approach can be either *static* or *dynamic* in nature. In the former case, the set of patterns is fixed and displayed in the GUI (e.g., Fig. 1). In the latter case, the patterns are dynamically suggested during visual query formulation by utilizing the knowledge of partially formulated query fragments. The AUTOG (Yi et al. 2017) framework adopts this dynamic strategy. Specifically, it au-

tomatically generates a small list of subgraph as suggestions by considering potential query result size as well as structural diversity. Figure 2 shows an example of subgraph suggestions during query formulation in AUTOG. Note that it is extremely difficult to speculate a priori the subgraph structure that a user wishes to construct during query formulation.

Lastly, recent frameworks such as VISAGE (Pienta et al. 2016) support *query by example* (QBE) for formulating graph queries using examples.

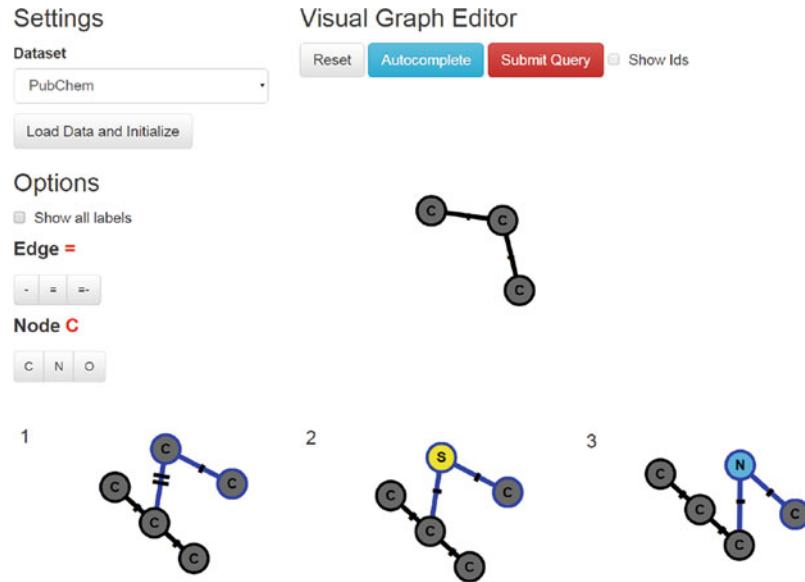
Recently, there has been another line of work that provides opportune *empty result feedback* during visual query formulation. Specifically, the goal is to detect during query construction whether a partially constructed query returns empty results or alerts users in a timely fashion so that one can undertake appropriate remedial action(s). Several recent studies (Bhowmick et al. 2015; Pienta et al. 2016) address this problem by notifying a user opportunely when a partially constructed visual query yields an empty result.



Visual Graph Querying, Fig. 1 Pattern-at-a-time visual query formulation

Visual Graph Querying,

Fig. 2 Subgraph suggestions during query formulation



Blending Visual Query Formulation and Processing

In traditional visual query processing paradigm, query evaluation can be performed in two key steps. First, the visual query is transformed into its textual or algebraic form. Second, the transformed query is evaluated using an existing state-of-the-art graph query processing method (Han et al. 2013). Observe that although the final query that a user intends to pose is revealed gradually in a step-by-step manner during query construction, it is not exploited by the query processor prior to clicking of the Run icon to execute the query. This often results in slower *system response time* (SRT), which is the duration between the time a user presses the Run icon and the time when the user gets the query results (Jin et al. 2012). This traditional view of visual graph query processing is primarily due to the fact that until recently the data management community has traditionally considered visual interface-related issues more relevant to the HCI community and orthogonal to data processing.

The techniques in Jin et al. (2010, 2012) and Hung et al. (2014) take a nontraditional step toward exploring a graph query processing paradigm by blending these two orthogonal areas. Specifically, it *interleaves* (i.e., *blend*) query construction and query processing to prune

false results and prefetch partial query results in a single-user environment by exploiting the availability of GUI *latency* (i.e., the time taken to construct an edge of a query graph visually) during visual query formulation. The key benefits of this paradigm are at least two-fold. First, it significantly improves the SRT (From an end user's perspective, the SRT is crucial as it is the time a user has to wait before she can view the results.) as the query processor does not remain idle during visual query formulation by processing the potential subgraph query early based on "hints" received from the user. Second, as a visual query is iteratively processed during query formulation, it paves way for realizing efficient techniques that can enhance usability of graph databases such as query suggestion and exploratory search (Huang et al. 2017).

This framework constructs offline and online indexes based on underlying features of data graphs. When a user adds a new edge e to a visual subgraph query q , it constructs and maintains an adaptive online index for the edge to facilitate blending of visual query formulation and processing. If the user is interested in exact subgraph matches, then it retrieves the candidates of q (stored in R_q) by leveraging the offline and online indexes. If R_q is empty, then it means that there is no exact match for q after the

addition of e . Consequently, the user can either modify q or retrieve similar matches to q . If the user chooses the latter, then q is regarded as a subgraph similarity query, and corresponding candidate matches are retrieved by leveraging the indexes. On the other hand, if the user chooses the former, then a user-selected edge is removed and the online index is updated. If the user clicks the Run button, then the constructed query q is processed to retrieve result matches. If q is an exact subgraph query, the exact results will be returned after conducting candidates verification (i.e., subgraph isomorphism test), if necessary, on R_q . Otherwise, if it is already a subgraph similarity query, then results that match q approximately are returned to the user.

Note that in the above framework, the expensive candidate verification step is performed only after the Run button is clicked and not during visual construction of the query fragments. Second, it allows a user to execute a query fragment anytime during query formulation and not wait until the entire query is visually formulated. This facilitates exploratory search as a user may formulate a partial query, execute it, browse the results, and then continue expanding it (Huang et al. 2017).

Interactive Exploration of Query Results

The preceding subsection highlights frameworks that exploit human interactions with a visual graph query interface during query formulation to process subgraph queries iteratively. Naturally, it is imperative for such visual subgraph querying frameworks to enable efficient exploration and visualization of result matches of such subgraph queries. This is a challenging problem as it requires effective summarization and visualization of the content and structure of the matching subgraphs involved in a potentially large collection of query results. In this subsection, we briefly describe efforts reported in recent literature to this end.

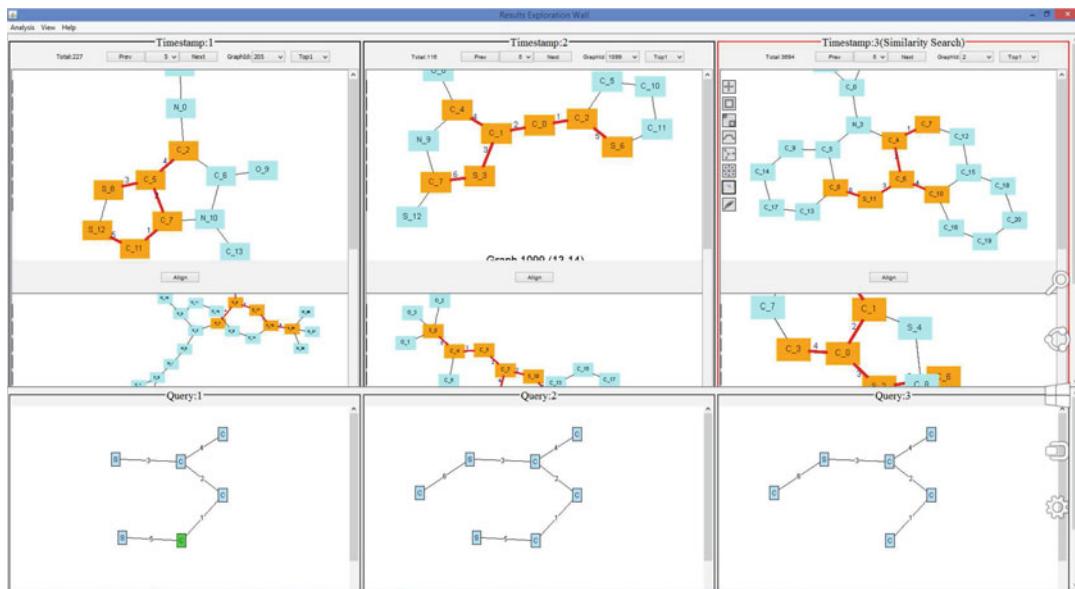
Despite the fact that exploration of query results is the first step toward sensemaking for a user, scant attention has been paid on this problem by both academic and commercial graph data management communities. Specifically, we

can categorize visual exploration of graph query results into two types: (a) visual exploration of query results in a large collection of small- and medium-sized graphs (Jin et al. 2010, 2012; Huang et al. 2017) and (b) visual exploration of result matches in a large network (Hung et al. 2014; Pienta et al. 2016, 2018). We elaborate on them in turn.

Exploration in a large collection of small- and medium-sized graphs. Early efforts for exploration of query results (Jin et al. 2010, 2012) simply displayed result matches in a list, without revealing connections among results. A matching subgraph in each data graph is then highlighted with different colors to identify the component of the data graph that matches a visual query. These results may be further sorted by various measures such as subgraph distance (for approximate match). For instance, in Jin et al. (2010, 2012), a user can only iteratively scroll through each result data graph to view query results. Clearly, this is tedious even for modest-sized query results and cannot easily reveal patterns and relationships between matching subgraphs.

PICASSO (Huang et al. 2017) is a system that utilizes the query processing engine described in the preceding subsection to support visual exploratory search. It accommodates the results of the initial and reformulated query graphs to be juxtaposed in the form of *parallel search streams* (i.e., parallel query-results pairs) that facilitate exploration of the underlying data and possible identification of new search directions. Figure 3 depicts an example of three parallel search streams during exploratory search in PICASSO. Furthermore, it provides a framework to further search and analyze various features of the search results during the exploration process to facilitate understanding of the data.

Exploration of result matches in a large network. It is well known that visualizing large graphs containing thousands of vertices and edges is cognitively challenging. Consequently, the aforementioned approach of highlighting the result matches of a visual subgraph query by color-coding them in the original data graph is



Visual Graph Querying, Fig. 3 Parallel search streams in PICASSO

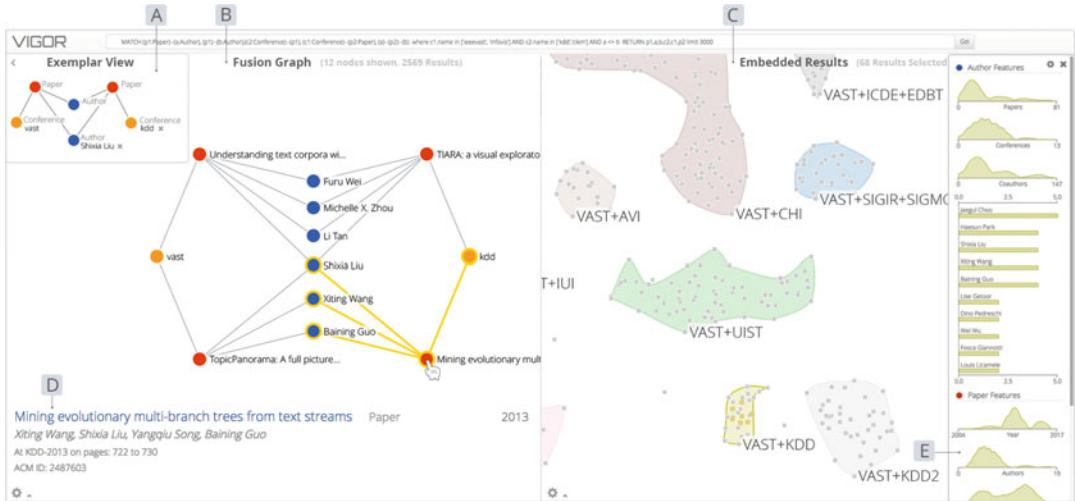
ineffective in the context of large networks as it is not only challenging to locate query results in a giant “hair ball” but also it is extremely difficult to comprehend the structural relationships among the vertices in a matching subgraph overlaid on a large network. Despite these challenges, there are very few works on query results exploration on large networks. Similar to Jin et al. (2010) and Jin et al. (2012), early efforts such as VISAGE (Pienta et al. 2016) simply display the results in form of a list.

Recently, there has been an increasing attention to develop advanced techniques for query results exploration for large networks. This work can be broadly classified into three types, namely, *region-based*, *exemplar-based*, and *feature-based exploration*. Intuitively, a *region-based exploration* scheme iteratively displays a small region of the underlying network containing a result match of a subgraph query. By showing only a fragment of the original network one at a time, it alleviates the cognitive overhead associated with the visualization of all query results on the original network. The approach in Hung et al. (2014) adopts this strategy. In an *exemplar-based exploration* scheme, a user can select a specific query result (i.e., an exemplar)

and relax its constraints to retrieve other similar results. A user can also start the exploration by specifying only the topology (without constraints on node values) and iteratively add constraints to narrow in on specific results. *Feature-based exploration* schemes, on the other hand, take a top-down approach by generating a high-level overview of all the query results. Specifically, it groups the query results based on the structural features and embeds them in a low-dimensional representation. VIGOR (Pienta et al. 2018) supports both exemplar-based and feature-based exploration schemes (Fig. 4).

An Example

Consider a chemical compound repository containing a large collection of small- or medium-sized graphs such as *PubChem*. Suppose a user wishes to formulate the subgraph query shown in Fig. 1. She may visually formulate it either using an interface that supports edge-at-a-time query formulation by adding the five edges iteratively or using a static pattern-at-a-time interface (e.g., Fig. 1) to quicken the formulation by dragging and dropping canned patterns. Alternatively, a



Visual Graph Querying, Fig. 4 Visual interface of VIGOR (Pienta et al. 2018)

dynamic pattern-at-a-time interface such as Fig. 2 can be leveraged where suggestions are generated iteratively as the user formulates her query. Specifically, Fig. 2 depicts the three suggestions generated by AUTOG (Yi et al. 2017) after the addition of two C-C edges. The query is shown in the middle of the GUI, whereas relevant suggestions are presented in the bottom. She may adopt the second suggestion and continue with the query formulation task. Next, AUTOG presents the user’s final query as Suggestion 3 as shown in Fig. 5. Consequently, the selection of this suggestion completes the query formulation task.

The techniques proposed in Jin et al. (2010, 2012) can be utilized to interleave aforementioned query formulation with query processing. Specifically, the query can be iteratively executed during formulation, and corresponding results can be explored to gain insights. Figure 3 depicts such exploration. The original query is shown in the bottom left panel. The modifications to this query with addition and deletion of edges are shown in the bottom middle and right panels, respectively. The corresponding search results for each of these subgraph queries are shown in the top. A user can compare the search results and explore the differences between the result sets and features of the nodes in them using PICASSO (Huang et al. 2017).

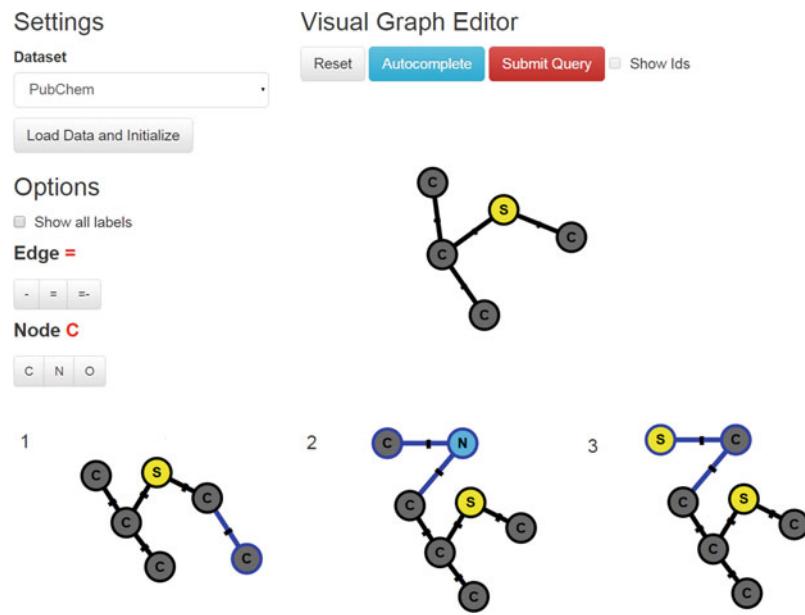
Future Directions

While good progress has already been made, research on visual graph querying opens up many opportunities for continued research as highlighted below. Some of these topics were introduced by recent vision papers (Bhowmick 2014; Bhowmick et al. 2016).

Data-driven construction of visual query interfaces. Most of the real-world GUIs for visual graph querying are constructed and maintained *manually*. That is, details of visual design of a GUI are manually worked out, and contents of various components are created manually by “hard coding” them during GUI implementation. Unfortunately, such manual effort may create GUIs that do not provide sufficient features to aid efficient query formulation, are static in nature when the underlying graph data repository evolves, and have limited portability (Bhowmick et al. 2016). To alleviate these limitations, a recent work introduced the paradigm of *data-driven* GUI construction and maintenance (Zhang et al. 2015), where the goal is to automatically construct the content of various panels of a GUI and maintain them as underlying data evolves. Such a data-driven paradigm has several benefits such as superior support for visual subgraph

Visual Graph Querying.

Fig. 5 Query suggestions of Suggestion 2 of Fig. 2, where Suggestion 3 is the completed query



query construction, significant reduction in the manual cost of maintaining an interface for any graph query-based application, and portability of the interface across diverse variety of graph querying applications. However, these are very early efforts, and there are many opportunities to explore this novel paradigm of GUI construction further.

Visual querying on massive graphs. All researches related to guidance for visual query formulation, visual action-aware query processing, and exploration and visualization of query results have focused either on a large set of small- or medium-sized data graphs or on networks with millions of nodes. A natural extension to this paradigm is to support similar problems on massive graphs (comprising hundreds to billions of nodes), which may demand a distributed framework and novel algorithms built on top of it.

Efficient processing of complex graph queries. Current research demonstrates the viability of blending visual formulation and processing of subgraph containment and subgraph similarity search queries. It is an open problem to enhance the expressiveness of such visual querying

framework to handle more complex subgraph queries such as homomorphism-based subgraph queries (Fan et al. 2010).

Multifaceted exploration and visualization of query results. As remarked earlier, techniques that enable rich, interactive exploration and visualization of graph query results are still in its infancy. *How can we easily explore and visualize results of a variety of subgraph queries to gain better understanding of it?* This is especially a challenging problem when the underlying graph is massive as the entire graph looks like a giant hair ball and the subgraphs that are returned as results to a query are lost in the visual maze. Furthermore, it is interesting to explore additional insights that we may attach to the matched results that may enable end users for further exploration.

Automated performance study. User studies are *sine qua non* for evaluating the performance and effectiveness of the aforementioned visual action-aware query processing techniques. In contrast to the traditional query processing paradigm where the runtime performance of a large number of subgraph queries can be easily measured by automatically extracting

a random collection of subgraphs from the underlying data and executing them (Katsarou et al. 2015), each visual query graph *must* be formulated by a set of users. This is because in this paradigm the availability of the GUI latency at each formulation step is exploited to prefetch and refine candidate matches. To address this challenge, it is important to automatically generate many test subgraph queries having different *user-specified characteristics* (e.g., frequent, infrequent) using indexes and simulate their formulation based on different query formulation sequences *without requiring human users*. A recent effort (Bhowmick et al. 2017b) addresses this by building an HCI-based framework for simulating subgraph query construction on a database containing a large number of small- or medium-sized graphs. It can be used to automate exhaustive evaluation of performances of various visual action-aware query processing techniques. However, this area is still in its infancy with only one notable work.

Cross-References

- ▶ [Graph Exploration and Search](#)
- ▶ [Graph Pattern Matching](#)
- ▶ [Graph Query Languages](#)

References

- Bhowmick SS (2014) DB \bowtie HCI: towards bridging the chasm between graph data management and HCI. In: Proceedings of the 25th international conference on database and expert systems applications (DEXA), Part I, Munich, 1–4 Sept 2014, pp 1–11
- Bhowmick SS, Dyreson CE, Choi B, Ang M (2015) Interruption-sensitive empty result feedback: rethinking the visual query feedback paradigm for semistructured data. In: Proceedings of the 24th ACM international conference on information and knowledge management (CIKM), Melbourne, 19–23 Oct 2015, pp 723–732
- Bhowmick SS, Choi B, Dyreson CE (2016) Data-driven visual graph query interface construction and maintenance: challenges and opportunities. PVLDB 9(12):984–992
- Bhowmick SS, Choi B, Li C (2017a) Graph querying meets HCI: state of the art and future directions. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD conference 2017, Chicago, 14–19 May 2017, pp 1731–1736
- Bhowmick SS, Chua H, Choi B, Dyreson CE (2017b) VISUAL: simulation of visual subgraph query formulation to enable automated performance benchmarking. IEEE Trans Knowl Data Eng 29(8):1765–1778
- Fan W, Li J, Ma S, Wang H, Wu Y (2010) Graph homomorphism revisited for graph matching. PVLDB 3(1):1161–1172
- Han W, Lee J, Lee J (2013) Turbo_{ISO}: towards ultrafast and robust subgraph isomorphism search in large graph databases. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD), New York, 22–27 June 2013, pp 337–348
- Huang K, Bhowmick SS, Zhou S, Choi B (2017) PICASSO: exploratory search of connected subgraph substructures in graph databases. PVLDB 10(12):1861–1864
- Hung HH, Bhowmick SS, Truong BQ, Choi B, Zhou S (2014) QUBLE: towards blending interactive visual subgraph search queries on large networks. VLDB J 23(3):401–426
- Jin C, Bhowmick SS, Xiao X, Cheng J, Choi B (2010) GBLENDER: towards blending visual query formulation and query processing in graph databases. In: Proceedings of the ACM SIGMOD international conference on management of data, (SIGMOD), Indianapolis, 6–10 June 2010, pp 111–122
- Jin C, Bhowmick SS, Choi B, Zhou S (2012) PRAGUE: towards blending practical visual subgraph query formulation and query processing. In: IEEE 28th international conference on data engineering (ICDE), Washington, DC, 1–5 Apr 2012, pp 222–233
- Katsarou F, Ntarmos N, Triantafillou P (2015) Performance and scalability of indexed subgraph query processing methods. PVLDB 8(12):1566–1577
- Pienta R, Tamersoy A, Endert A, Navathe SB, Tong H, Chau DH (2016) VISAGE: interactive visual graph querying. In: Proceedings of the international working conference on advanced visual interfaces (AVI), Bari, 7–10 June 2016, pp 272–279
- Pienta R, Hohman F, Endert A, Tamersoy A, Roundy K, Gates C, Navathe SB, Chau DH (2018) VIGOR: interactive visual exploration of graph query results. IEEE Trans Vis Comput Graph 24:215–225
- Tian Y, Patel JM (2008) TALE: a tool for approximate large graph matching. In: Proceedings of the 24th international conference on data engineering (ICDE), Cancún, 7–12 Apr 2008, pp 963–972
- Yan X, Yu PS, Han J (2004) Graph indexing: a frequent structure-based approach. In: Proceedings of the ACM SIGMOD international conference on management of data, Paris, 13–18 June 2004, pp 335–346
- Yan X, Yu PS, Han J (2005) Substructure similarity search in graph databases. In: Proceedings of the ACM SIGMOD international conference on management of data, Baltimore, 14–16 June 2005, pp 766–777
- Yi P, Choi B, Bhowmick SS, Xu J (2017) Autog: a visual query autocompletion framework for graph databases. VLDB J 26(3):347–372

Zhang J, Bhowmick SS, Nguyen HH, Choi B, Zhu F (2015) Davinci: data-driven visual interface construction for subgraph search in graph databases. In: 31st IEEE international conference on data engineering (ICDE), Seoul, 13–17 Apr 2015, pp 1500–1503

Visualization

Ahmed Eldawy

Department of Computer Science and Engineering, University of California, Riverside, CA, USA

Synonyms

Cartography

Definitions

Visualization of big spatial data is the process of generating a pictorial representation of a spatial dataset. What signifies spatial data visualization is that the spatial attributes in the data, e.g., lines and polygons, are mainly used to produce the generated image which results in a wide range of possible visualizations.

Overview

Visualization is one of the most important features used with spatial data processing as it gives a bird's-eye view of the data. It is a natural way to browse and interact with spatial data. As the spatial data gets bigger, interactive visualization of such data is a paramount need for scientists in various domains. It helps in identifying interesting patterns, anomalies, or special conditions which are usually hard to detect. For example, biochemists use visualization to predict protein structure (Cooper et al. 2010), planetary scientists found ice on Mars by visualizing thermal images (Smith et al. 2016), neuroscientists

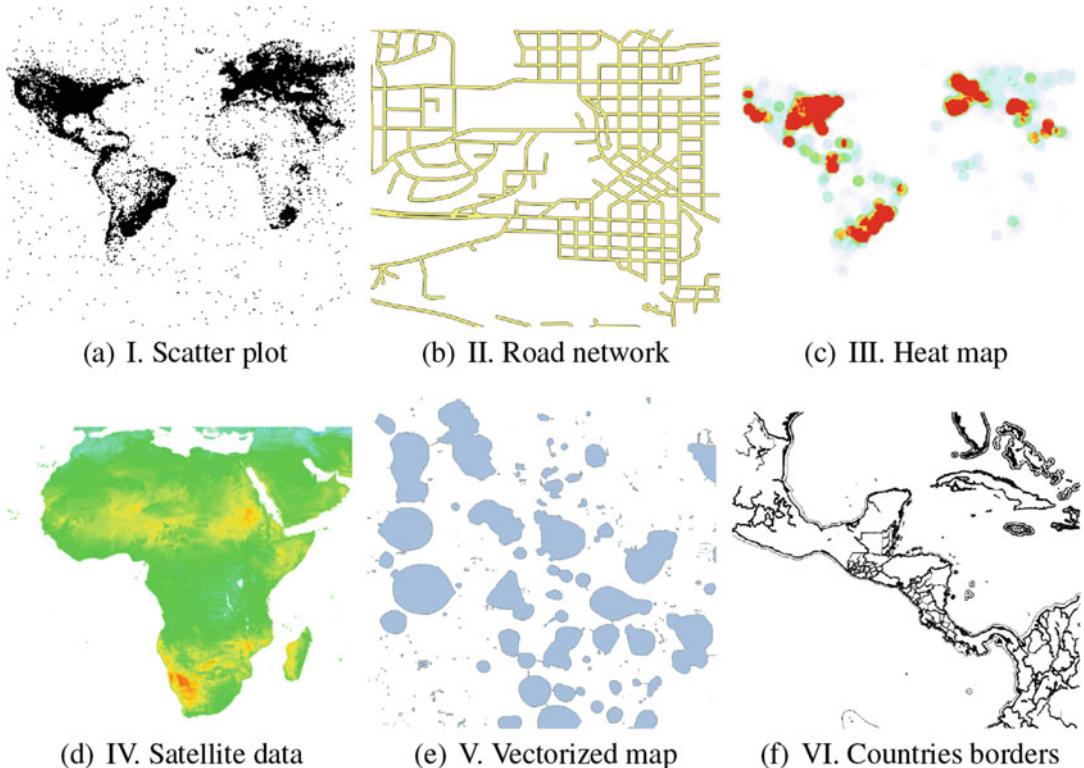
study the functions of the brain via interactive visualization of simulated data (Markram 2006), scientists in the field of connectomics analyze 3D microscopic images to understand the structure of the nervous system (Lichtman et al. 2014), and doctors discover new knowledge by visualizing electronic health records (EHR) (West et al. 2015).

While nonspatial data visualization is usually concerned with aggregating the data to produce bar charts or pie charts, spatial data visualization is more involved as it produces images that describe the input records. It can be as simple as a scatterplot or a heat map for a set of points. But it can also be used to produce more sophisticated images such as visualizing a road network, vehicle trajectories, brain models (Markram 2006), and protein structure (Cooper et al. 2010). Figure 1 portrays a few examples of spatial data visualization.

Traditionally, spatial data visualization was performed using single-machine algorithms that keep all the data in the main memory (Middel 2007; Cruz et al. 2013; Song et al. 2012; Maciejewski et al. 2010; Bezerianos et al. 2010; Elmquist et al. 2008). These techniques focus on defining how the generated image should look like per the application needs, while the performance was out of scope. As the data gets bigger, there is a dire need to improve the performance of the traditional techniques to scale to terabytes of data. However, there are many challenges that have to be addressed in big spatial data visualization as described below.

Data Size: Traditional techniques were designed for small datasets that can fit in the main memory. These techniques were inherently sequential and cannot be easily parallelized to big data.

Abstraction: In order to handle big datasets, parts of the visualization process have to be injected in big data platforms, such as Hadoop or Spark. The challenge is to identify the minimal and representative set of abstract operations that can be parallelized to serve a wide range of visualizations.



Visualization, Fig. 1 Examples for spatial data visualization. (a) I. Scatter plot. (b) II. Road network. (c) III. Heat map. (d) IV. Satellite data. (e) V. Vectorized map. (f) VI. Countries borders

Big Images: Big spatial data visualization is not only about big data but also about big images. As the size of the data increases, there are more details to encode in the generated image which requires producing terapixel images that catch as much details as the users want.

Interactivity: Since big spatial data visualization is about data exploration, the generated visualization has to provide interactive exploration capabilities where users can zoom in and out or pan around the produced image.

The research in big spatial data visualization can be broadly categorized in three categories, *visualization abstraction*, *single-level visualization*, and *multilevel visualization*. The visualization abstraction focuses on defining a descriptive and powerful abstraction that can be used to produce a wide range of visualizations. The single-level visualization is concerned with the generation of a fixed-resolution image that describes

the underlying data with no interactivity such as zoom in and out. Multilevel visualization aims to produce a multilevel image where users can interactively zoom in and out to see more or less details. The rest of this article describes the state-of-the-art research in each of these three topics.

Key Research Findings

Visualization Abstraction

The visualization abstraction defines a set of core functions that can be reused and/or customized to build a wide range of visualizations. In nonspatial data visualization, these abstractions include sampling, aggregation, and filtering (Wu et al. 2014; Battle et al. 2013; Jugel et al. 2014; Wesley et al. 2011). Defining these abstractions allowed some systems to handle big data visualization by injecting these functions inside a database system or a big data platform to scale up to terabytes of

data (Wu et al. 2014; Battle et al. 2013). Unfortunately, these abstractions are not very descriptive when it comes to spatial data visualization such as the visualization of trajectories or road networks.

Protopvis (Bostock and Heer 2009), Vega (Satyanarayan et al. 2016, 2017), and D3 (Bostock et al. 2011) provide more sophisticated visualization abstractions that can be used to produce complex visualizations. They allow users to link data sources to visualizations to produce nice and complex visualizations. In particular, Protopvis focuses on building complex visualizations linked to the input data with the goal of producing more complicated visualizations. D3 is concerned with integrating the visualizations with the data object model (DOM) used in HTML documents to produce web-ready visualizations. Vega aims to add interactivity to these visualizations by allowing the users to define *selectors* that can filter the data being visualized. These systems provide good examples of different visualization abstractions used in various applications. They focused on being descriptive rather than scaling to terabytes of data.

HadoopViz (Eldawy et al. 2016), on the other hand, provides an abstraction that focuses on the scalability. Rather than decomposing the produced image into primitive elements, HadoopViz focuses on data partitioning and image partitioning where small subsets of the data can be processed independently to produce parts of the final image. Then, the partial images are combined together to produce the final image. Unfortunately, this abstraction does not allow any interactivity as the produced image is not linked to the original data used to produce the image. Nonetheless, it can scale to terabytes of data by seamlessly allowing distributed processing using the Hadoop MapReduce platform.

Single-Level Visualization

This part describes the recent research on producing a single-level image with a fixed resolution. A single-level image is a regular image, e.g., PNG, which provides a pictorial representation of a dataset. Notice that the produced single-level image can be in a vector format, e.g., SVG, but it still has a fixed resolution, or level of details,

which is embedded in the generated image. If the user wants a higher or a lower resolution, the image has to be regenerated.

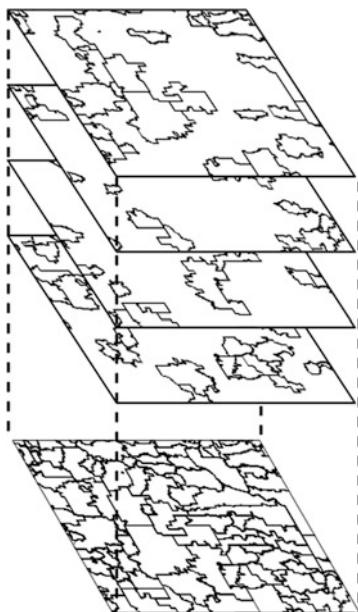
When it comes to big spatial data visualization, the main challenge is how to process all the input data in parallel to produce the final image. The basic idea is to partition the data, allow each machine to visualize each partition to produce a partial image, and finally combine the partial images to produce the final image. Depending on how the data is partitioned, single-level visualization techniques can be categorized into three categories, nonspatial partitioning, pixel-level partitioning, and spatial partitioning.

Nonspatial Partitioning

In nonspatial partitioning, the data is partitioned using any traditional partitioning method, e.g., random, round-robin, or hash partitioning. By default, most big data platforms, e.g., Hadoop and Spark, use a block-based random partitioning where each fixed-size block of data (128 MB) is assigned to three machines chosen at random. As a result of this nonspatial partitioning, each block or partition might contain data from the entire input space. Therefore, when a partition is visualized, the produced partial image will be mostly as big as the final image as it covers the entire input space (Eldawy et al. 2016).

Figure 2 shows an example of visualizing a set of polygons that represent the boundaries of ZIP codes. Each partition contains a set of polygons that cover the entire input space. In order to combine them into one final image, we *overlay* the partial images on top of each other. This requires defining a function that takes the individual pixel values from the partial images and produce one pixel value (or color) in the final image. One example is to initialize each partial image with a transparent background that reveals the corresponding pixels in other partial images as shown in Fig. 2.

The advantage of nonspatial partitioning is that the partitioning step is typically very fast. In many cases, the input data is already partitioned in a distributed file system, such as HDFS, and the partitioning step can be skipped altogether. The drawback is that as the desired image size



Visualization, Fig. 2 Visualization using nonspatial partitioning. The partial images are overlaid to produce the final image

increases, the sizes of partial images become too large and might slow down the overlay step and the entire visualization process. For example, if a giga-pixel image is desired, each partial image will contain a few gigabytes of data that need to be transferred over network to combine in a single machine that will finally process all of them to produce the final image.

Pixel-Level Partitioning

Pixel-level partitioning tries to resolve the drawback of nonspatial partitioning by carefully partitioning the data such that all the data records that affect a single pixel are combined together in one partition. For example, when visualizing satellite data, each pixel in the produced image can be mapped to a region on the Earth surface. If all the data in that region are combined together in one partition, then we can produce the final color of that specific pixel without considering any other data. This technique was used to visualize 3D meshes (Vo et al. 2011) and satellite data (Planthaber et al. 2012).

In pixel-level partitioning, the number of partitions can be as large as the number of pixels

in the desired image. For example, to generate a 1000×1000 pixels image, up to one million partitions will be created. Notice that the number of partitions can be smaller in the case when some partitions do not contain any data, e.g., water areas when visualizing land data. While a huge number of partitions can be created, this technique has two main advantages. First, there is no need to define a logic that combines different pixel values into one final value as each pixel is generated by exactly one partition. Second, regardless of the number of partitions, the output of each partition is always a single pixel value which allows this technique to scale to larger images than nonspatial partitioning techniques.

Spatial Partitioning

This technique is a generalization of the pixel-level partitioning. Instead of creating exactly one partition for each pixel, this technique partitions the input data into regions such that each region covers a rectangular region of the final image which might contain more than one pixel. The spatial partitioning takes into account the data distribution so that each partition contains roughly the same amount of data. Each partition is then processed to produce a partial image that contains only the pixels covered by that partition. The partial images are finally *stitched* together to produce one final image (Eldawy et al. 2016).

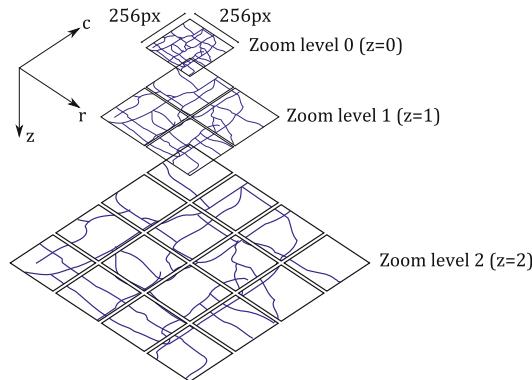
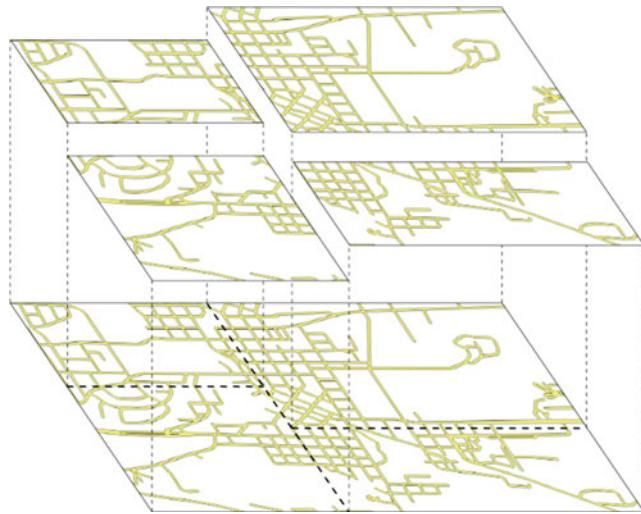
Figure 3 gives an example of visualizing road segments where partial images are stitched together, like a jigsaw puzzle, to produce the final image. This technique can reduce the number of partitions as compared to pixel-level partitioning while still maintaining its advantages. While each partial image can be larger than a single pixel, they are still much smaller than the entire image. If a uniform grid partitioning is used where the number of grid cells is equal to the number of pixels in the output image, it transforms to the pixel-level partitioning technique.

Multilevel Visualization

Figure 4 shows an example of a multilevel image with three levels, also called a three-level pyramid. Unlike a flat single-level image, the multilevel image consists of a large number of

Visualization, Fig. 3

Visualization using spatial partitioning. The partial images are stitched to produce the final image



Visualization, Fig. 4 A multilevel image with three levels

fixed-size tiles arranged in a pyramid structure. The tiles in each level collectively cover the entire input space. Each tile is identified by the tuple (z, c, r) , where z is the zoom level and c and r are the column and row number of the tile in the grid of level z , respectively. As we go deeper in the image, the number of tiles increases; hence, the resolution of the image also increases.

Multilevel visualization processes an input data to produce a multilevel image with a user-specified number of levels and tile resolution. The challenge is to produce all the tiles efficiently. Notice that the number of tiles increases exponentially with the number of zoom levels. For a multilevel image with 10 levels, more than a million tiles might need to be generated. There

are two techniques for multilevel visualization based on the partitioning technique as detailed below.

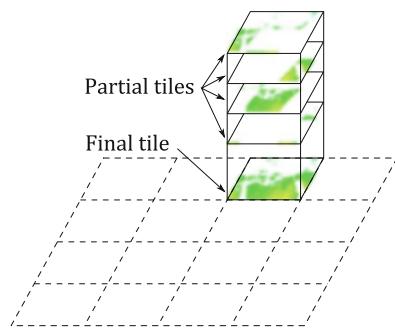
Nonspatial Partitioning

If a nonspatial partitioning is used, e.g., hash partitioning, then the records in each partition might span the entire input space. This means that each partition might contain records that cover *all* the tiles in *all* the zoom levels. Each partition is then processed independently to produce a multilevel image that is as big as the final image, i.e., it contains the same number of tiles as in the final image. However, these tiles are considered partial tiles and need to be merged to produce the final image for that particular tile.

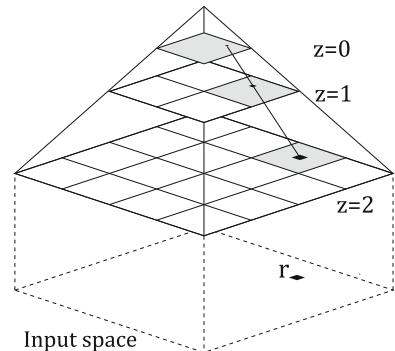
Figure 5 shows an example of how four partial tiles are merged to produce one final tile in the third level of a multilevel image. Similar to the nonspatial partitioning in single-level visualization, the partial tiles are overlaid to produce the final tile. While each tile has a fixed resolution, e.g., 256×256 pixels, there can be millions of tiles which might incur a huge overhead in the merge process.

Pyramid Partitioning

In pyramid partitioning, the data is partitioned according to the pyramid structure such that all the records that contribute to the visualization of one tile are grouped together in one partition.



Visualization, Fig. 5 Overlaying partial tiles in multi-level images



Visualization, Fig. 6 Pyramid partitioning

This means that each tile is generated by only one machine which completely eliminates the need of a merged step.

Figure 6 shows an example of pyramid partitioning when applied to a single record r . As shown in the figure, the record r is assigned to three partitions, one at each level. This means that the same record is processed three times or, in general, as many as the number of zoom levels in the multilevel image. While this technique eliminates the need of the merged step, it suffers from one major limitation; it has to process each tile on a single machine. If one tile overlaps with too many records, that single tile might be a bottleneck in the visualization process. At an extreme case, the single tile at the top of the pyramid always covers the entire input space. Therefore, visualizing that single tile using pyramid partitioning requires a single machine to process all the input dataset. To overcome this limitation, the nonspatial and pyramid partitioning techniques can be combined

together where the nonspatial partitioning is used to produce the first few levels, while the pyramid partitioning is used to visualize the deeper levels of the pyramid (Eldawy et al. 2016).

Examples of Applications

Visualization is used in many applications for interactive exploration. A major application is the visualization of maps which is also known as cartography (Kraak and Ormeling 2011). Biochemists use visualization to predict protein structure (Cooper et al. 2010). Planetary scientists found ice on Mars by visualizing thermal images (Smith et al. 2016). Neuroscientists study the functions of the brain via interactive visualization of simulated data (Markram 2006). Scientists in the field of connectomics analyze 3D microscopic images to understand the structure of the nervous system (Lichtman et al. 2014). Doctors discover new knowledge by visualizing electronic health records (EHR) (West et al. 2015).

Future Directions for Research

There are many open research problems in the area of big spatial data visualization. First, visualizing three-dimensional data is particularly challenging as the final image changes depending on the camera placement. This means that once an image is generated, the user cannot rotate the view without regenerating the entire image. Second, visualizing dynamic data interactively is also challenging as each new record invalidates the generated image. As the rate of record updates increases, there must be a way to partition the records across machines and update the generated image in parallel.

Cross-References

- ▶ [Big Data Visualization Tools](#)
- ▶ [Linked Geospatial Data](#)
- ▶ [Visualization Techniques](#)

References

- Battle L, Stonebraker M, Chang R (2013) Dynamic reduction of query result sets for interactive visualization. In: Proceedings of the 2013 IEEE international conference on big data, 6–9 Oct 2013, Santa Clara, pp 1–8. <https://doi.org/10.1109/BigData.2013.6691708>
- Bezerianos A, Chevalier F, Dragicevic P, Elmquist N, Fekete J (2010) GraphDice: a system for exploring multivariate social networks. *Comput Graph Forum* 29(3):863–872. <https://doi.org/10.1111/j.1467-8659.2009.01687.x>
- Bostock M, Heer J (2009) Protovis: a graphical toolkit for visualization. *IEEE Trans Vis Comput Graph* 15(6):1121–1128
- Bostock M, Ogievetsky V, Heer J (2011) D³: data-driven documents. *IEEE Trans Vis Comput Graph* 17(12):2301–2309
- Cooper S, Khatib F, Treuille A, Barbero J, Lee J, Beenen M, Leaver-Fay A, Baker D, Popovic Z, players F (2010) Predicting protein structures with a multiplayer online game. *Nature* 466(7307):756–760. <https://doi.org/10.1038/nature09304>
- Cruz IF, Ganesh VR, Caletti C, Reddy P (2013) GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics. In: 21st SIGSPATIAL international conference on advances in geographic information systems, SIGSPATIAL 2013, Orlando, 5–8 Nov 2013, pp 534–537. <https://doi.org/10.1145/2525314.2525324>
- Eldawy A, Mokbel MF, Jonathan C (2016) Hadoopviz: a mapreduce framework for extensible visualization of big spatial data. In: 32nd IEEE international conference on data engineering, ICDE 2016, Helsinki, 16–20 May 2016, pp 601–612
- Elmquist N, Dragicevic P, Fekete J (2008) Rolling the dice: multidimensional visual exploration using scatterplot matrix navigation. *IEEE Trans Vis Comput Graph* 14(6):1539–1148. <https://doi.org/10.1109/TVCG.2008.153>
- Jugel U, Jerzak Z, Hackenbroich G, Markl V (2014) M4: a visualization-oriented time series data aggregation. *PVLDB* 7(10):797–808. <http://www.vldb.org/pvldb/vol7/p797-jugel.pdf>
- Kraak MJ, Ormeling F (2011) Cartography: visualization of spatial data. Guilford Press, New York
- Lichtman JW, Pfister H, Shavit N (2014) The big data challenges of connectomics. *Nat Neurosci* 17:1448–1454
- Maciejewski R, Rudolph S, Hafen R, Abusalah AM, Yakout M, Ouzzani M, Cleveland WS, Grannis SJ, Ebert DS (2010) A visual analytics approach to understanding spatiotemporal hotspots. *IEEE Trans Vis Comput Graph* 16(2):205–220. <https://doi.org/10.1109/TVCG.2009.100>
- Markram H (2006) The blue brain project. *Nat Rev Neurosci* 7(2):153–160
- Middel A (2007) A framework for visualizing multivariate geodata. In: Visualization of large and unstructured data sets, pp 13–22
- Planthaber G, Stonebraker M, Frew J (2012) EarthDB: scalable analysis of MODIS data using SciDB. In: Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data, BigSpatial@SIGSPATIAL 2012, Redondo Beach, 6 Nov 2012, pp 11–19. <https://doi.org/10.1145/2447481.2447483>
- Satyanaarayan A, Russell R, Hoffswell J, Heer J (2016) Reactive Vega: a streaming dataflow architecture for declarative interactive visualization. *IEEE Trans Vis Comput Graph* 22(1):659–668
- Satyanaarayan A, Moritz D, Wongsuphasawat K, Heer J (2017) Vega-lite: a grammar of interactive graphics. *IEEE Trans Vis Comput Graph* 23(1):341–350
- Smith IB, Putzig NE, Holt JW, Phillips RJ (2016) An ice age recorded in the polar deposits of Mars. *Science* 352(6289):1075–1078
- Song J, Frank R, Brantingham PL, LeBeau J (2012) Visualizing the spatial movement patterns of offenders. In: SIGSPATIAL 2012 international conference on advances in geographic information systems (formerly known as GIS) (SIGSPATIAL'12), Redondo Beach, 7–9 Nov 2012, pp 554–557. <https://doi.org/10.1145/2424321.2424413>
- Vo HT, Bronson J, Summa B, Comba JLD, Freire J, Howe B, Pascucci V, Silva CT (2011) Parallel visualization on large clusters using MapReduce. In: IEEE symposium on large data analysis and visualization, LDAV 2011, Providence, 23–24 Oct 2011, pp 81–88. <https://doi.org/10.1109/LDAV.2011.609231>
- Wesley RMG, Eldridge M, Terlecki P (2011) An analytic data engine for visualization in tableau. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2011, Athens, 12–16 June 2011, pp 1185–1194. <https://doi.org/10.1145/1989323.1989449>
- West VL, Borland D, Hammond WE (2015) Innovative information visualization of electronic health record data: a systematic review. *J Am Med Inform Assoc* 22(2):330–339
- Wu E, Battle L, Madden SR (2014) The case for data visualization management systems. *PVLDB* 7(10):903–906. <http://www.vldb.org/pvldb/vol7/p903-wu.pdf>

Visualization Techniques

V

Rogério Abreu de Paula

IBM Research, Rua Tutóia 1157, São Paulo,
Brazil

Synonyms

Exploratory Data Analytics; Visual Data Exploration

Definitions

Visualization techniques for big data exploration, also known as *visual data exploration* (both terms will be used interchangeably in this article.), refer to a particular field of data visualization research that aims at devising and employing visualization techniques to aid the understanding of (extremely) large data sets (Keim 2001). These techniques evolved from the original concept of data visualization, which was originally defined as “the use of computer-supported, interactive, visual representations of abstract data to amplify cognition” (Card et al. 1999). The goal is thus to make information ready at hand so that people can make informed decisions in a timely fashion. The use of data visualization is often divided into two basic tasks (and tools thereof): data exploration and data representation. In *data exploration*, users are primarily concerned with investigating a data set in order to manipulate the data, learn more about it, test hypotheses, and the like. In *data representation*, on the other hand, users are primarily concerned with conveying specific information, highlighting particular properties and/or features of the data, and the like. Visual data exploration refers to the processes of achieving the former by means of the latter.

The visual exploration of big data poses unique technological challenges to the development of visualization techniques. Not only is the sheer size of big data a challenge but also its intrinsic characteristics, such as heterogeneity, high-dimensionality, and real-time nature. As Heer and Kandel (2012) put it, big data is tall (extremely large data sets), wide (high-dimensional data), and diverse (very heterogeneous data). Also, big data is fast (dynamic and time-sensitive). These characteristics directly affect the ability of visualization techniques to handle large data sets, often requiring data preprocessing and rendering techniques suitable for conveying appropriately massive amounts of data (Keim et al. 2006). These are long-lasting challenges in the field of information visualization (Chen 2005).

To enable users to extract meanings and gain insights from extremely large sets of data (i.e.,

big data), visualization techniques combine data processing and analysis, visualization techniques, and interactive visual interfaces. This refers to an emerging field known as *visual analytics*. Visual analytics is defined as the science and technique of analytical reasoning supported by interactive visual techniques and interfaces (Cook and Thomas 2005). Its methods and techniques allow decision-making over complex sets of data by leveraging on human’s inherent visual cognitive capabilities and modern interactive data processing technologies (Keim et al. 2008a). It supports a wide range of data processing tasks, but, in particular, it is conducive to visual (big) data exploration.

Overview

Everyday activities are deluged with data that has been growing exponentially in the past 20 years. But, data has no value in itself. It has to be processed, rendered, analyzed, and ultimately interpreted by human beings in order to create value. Approaches that enable them to explore and make sense of such large sets of complex data become paramount. This is not without challenges, notwithstanding. Such a data explosion creates the requirements for approaches that handle not only the sheer size but other intrinsic characteristics of big data, namely (Heer and Kandel (2012) outline the first three):

- Data is **tall**: extremely large data sizes that may contain millions or billions of records;
- Data is **wide**: a single data may contain hundreds or even thousands of features;
- Data is **diverse**: it may come from multiple data sources with varied data types; and,
- Data is **fast**: it may be highly dynamic and time-sensitive.

These characteristics make it rather difficult to extract meanings (and value thereof) from big data without the combination of advanced data analytics and interactive interfaces through which users can steer the exploration processes.

Hence, visualization techniques play a critical and central role in enabling big data exploration and discovery by means of integrating the increasing power of computer algorithms that analyzes extremely large data sets, the power of well-designed visual representations that make evident critical characteristics and relationships of the underlying structures of data, and the human cognitive power that is innately capable of identifying patterns, relationships, and the like from complex visualizations.

Big data exploration can be thus thought of as a process through which users investigate large data sets in attempts to identify useful information and gain insights, often without knowing beforehand what to look for Bikakis et al. (2016). It is an iterative process whereby users continuously process data, gather more information, select and combine different data representations, and incrementally gain new insights to answer a question at hand. To this end, visualization techniques should enable users to manipulate complex data sets in order to identify clusters, gaps, outliers, trends, and relationships in their underlying (often hidden) structures.

According to Thomas and Cook (2006), the aims of visualization techniques for data exploration are as follows:

- To increasingly facilitate understanding of massive and ever-growing collections of heterogeneous data sets;
- To support various data types, such as spatial and temporal data (in addition to unstructured data formats);
- To support understanding of uncertain, incomplete, and even misleading information;
- To enable users to integrate the context of their information needs into the data exploration processes; and
- To support multiple levels of data abstraction, summarization, and integration of data of various formats.

Yet, the effectiveness of these techniques rests on human inherent visual perception and reasoning abilities (i.e., its interpretability). These cog-

nitive capabilities enable humans to detect and recognize visual patterns, such as shapes, sizes, textures (and colors), distances, similarities, and more, effortlessly. Humans have an innate ability of understanding complex information visually (Thomas and Cook 2006). In contrast, human symbolic manipulation capabilities (e.g., dealing with numbers) require greater cognitive effort (and training). Contrary to common wisdom, numbers do not always speak for themselves. The goal of visualization is thus to “show the numbers,” that is, how to best represent and convey numerical (or categorical) information so that one can best extract meanings and values out of data and thus make the most appropriated informed decision.

In this context, many factors affect the effectiveness of visualization techniques, such as:

- **Screen attributes** determine not only the kind and amount of information that can be displayed but the choices of the most appropriate visual metaphor (Bieh-Zimmert and Felden 2014) and the ways in which users will be able to interact with the visualization. Key screen attributes are size, pixel, density, resolution, brightness, contrast, viewing angle, bezels, display technologies, and form factor (Andrews et al. 2011). They do not equally affect the ability of displaying and making sense of large data sets. For instance, the number of data elements in big data scenarios by far exceeds the number of pixels of even ultrahigh-resolution displays. This would lead to overplotting if one attempted to visualize every data point.
- **Visual metaphor** defines the ways in which data is represented visually. The question is to determine the most appropriate visual metaphor (or their combination thereof) that can best convey the intrinsic characteristics of the data being explored. A critical task is to identify the right metaphor that can best identify gestalt (see Human perception below), convey meanings, facilitate discovery and interpretation, enhance detection and recognition, and the like (Meirelles 2013).

- **Interactivity** is the mechanism by which users can manipulate and explore large data sets by means of selecting, filtering, zooming, brushing, scaling, and panning the data. Doing so, users can overview the data, select particular perspective(s) by zooming and filtering, and drill down to any particular area of interest. In addition, users may select different color palettes to best represent different data characteristics, different metaphors, and the like in order to make sense of the data. To coordinate actions across multiple representations is also critical to enable users to explore more complex, multidimensional data sets.
- **Data preprocessing** refers to the use of data analysis algorithms in order to reduce the amount and determine the kind of information to be rendered and displayed. In many visualization techniques, the direct mapping between data points and visual items results in occlusion and cluttering (Liu et al. 2013). Data analytics enables users to visualize just the “right” data. It is comprised of techniques and algorithms for aggregating, clustering, filtering, sorting, and the like the data.
- **Human perception** is a critical factor in determining the effectiveness of a visualization technique for big data exploration. Human’s innate cognitive capability of identifying visual patterns or outliers in images, as described in Gestalt psychology, plays a critical role in enabling visual exploration of large data sets. For instance, linearities and nonlinearities, continuities and discontinuities, clusters and outliers, and grouping and distinctions are important Gestalt features that affect how humans perceive and consequently make sense of visual data (Buja et al. 1996).

Visualization Techniques for Big Data Exploration

Visualization techniques for data exploration are a means for making information ready at hand so that people can make the most informed decisions in a timely fashion. And, many are those

techniques, as aforementioned. They primarily refer to a set of algorithms and approaches that convert data into visual structures based on different visual encodings so that it can be rendered and displayed. Those encodings depend on the nature of the data (i.e., its structure and composition). Some data will be best represented by pixel-oriented methods (e.g., dense pixel), icon-based methods (e.g., “glyphs”), axis-based methods (e.g., scatterplot matrices, parallel coordinates), graph-oriented methods (e.g., network data), or their combination.

While the next sections will cover individually the most critical visualization techniques for data exploration, it is noteworthy that most of the practical visual data exploration solutions employ various combinations of these techniques as well as others. Significantly, there are many ways in which these techniques can be “snapped together” in support of exploration tasks, such as **linking and brushing** (Becker and Cleveland 1987; Ward 2009), **panning and zooming**, **focus-plus-context**, and the like. The overall idea is to interconnect and coordinate the actions (e.g., selecting, filtering, zooming, etc.) on one visualization across all other available ones. In sum, multiple coordinated visualization techniques are critical to enable complex data exploration (North and Shneiderman 2000).

Pixel-Oriented Methods

Pixel-oriented (Keim 2000) or **dense pixel** (Keim 2001) techniques render each data point as a colored pixel where each dimension value is represented by a particular color and all pixels that are part of the same dimension are grouped into adjacent areas. In so doing, they in general partition the screen in N subarea, where N is the number of dimensions (or attributes/features) of the data. The resulting visualization allows users to identify local correlations, dependencies, and key areas within and across dimensions (or data attributes). These techniques aim at supporting the visualization and exploration of large amounts of multidimensional data. These techniques tend to suffer from the problems of occlusion and cluttering when attempting to display very large data sets (Liu et al. 2013).

Icon-Based Methods

Icon-based techniques (Nocke et al. 2005) render each data item as an icon and data attributes (or dimensions) as features of the icons. This is a proved approach for visualizing multivariable or multidimensional data, where data variables are mapped onto geometric (e.g., shape, size, orientation, position) and nongeometric (e.g., color, texture) visual attributes. Those visual “icons” are also known as “glyphs” (Ward 2002), which are defined as graphical entities that display one or more data values by means of those visual attributes (see Ward 2008 for a thorough description of how to generate and organize effective glyphs for multivariate data). The value of this approach stems from effectively communicating data attributes or supporting the detection of similarities, differences, clustering, outliers, and relationships in multivariable data. Certain mapping choices between data attributes and visual attributes might lead the users to privilege some relationship over others because of perception biases. It is thus critical to select the attributes in a systematic way as well as to employ techniques for reducing clutter and to order the attributes effectively.

Axis-Based Methods

Axis-based methods represent data item dimensions, and/or variables are represented as axial relationships in a 2D plane (Liu et al. 2015). The most well-known visualization techniques of this class are parallel coordinates and scatterplot matrices.

Parallel coordinates techniques are non-projective mapping between N -dimensional and two-dimensional sets (Inselberg and Dimsdale 1987, 1990). In essence, it differs from other multidimensional visualization techniques by representing the relations rather than finite point sets. On a two-dimensional plane with xy-Cartesian coordinates, the N -dimensional space is represented as N parallel and equidistant lines perpendicular to the x-axis. A point C with coordinates (c_1, c_2, \dots, c_N) is represent as a polygonal lines whose i -vertex is at c_i distance from x-axis for $i = 1, \dots, N$. This

technique allows multivariate relation patterns to be revealed by showing all dimensions simultaneously. Liu et al. (2015) point out that since this is essentially a linear ordering of N parallel lines in the x-axis, there are $N!$ possible permutations. Hence, the greatest challenge is to determine the most appropriate order of those features. Overplotting and visual clutter also become an issue when the number of points to be represented becomes too big. A data technique, called **focus+context**, has been employed to reduce clutter while at the same time highlighting key features in a parallel coordinate plot (Novotny and Hauser 2006), where both trends and outliers are captured and treated appropriately: trends that capture the overall relationship between axes are rendered as regions (polygon strips) and outliers are rendered as single lines. This is also an interactive technique in that selected data items are visually highlighted (line or regions of different colors).

Scatterplot matrix techniques (Carr et al. 1987) represent N dimensions as $N(N-1)$ ordered bivariate scatterplots. They are comprised of a collection of scatterplots, each of which representing the relationship between two dimensions (or features). In essence, they allow users to view multiple bivariate relationships simultaneously (Liu et al. 2015). For large data sets, the goal of each scatterplot is to represent data density and the matrix to allow the comparison of those density distributions across dimensions. Scatterplots are axis-based techniques for depicting the relationship between two variables/dimensions. The density representation could employ different visualization artifacts, such as color palettes, gray scale, and shapes (e.g., hexagons, circles, and the like). For very large data sets, visual cluttering is a common problem with scatterplots as many values may appear in the same location – though high-resolution displays may help to alleviate the overplotting problem (Andrews et al. 2011). In addition, since the number of bivariate scatterplots increases quadratically with the increase in the number of dimensions, approaches have been introduced to automatically or semiautomatically identify or rank the most interesting ones (see Liu et al. 2015).

Graph-Based Methods

Methods for graph visualization essentially aim at rendering relational data (i.e., data that is structured as nodes and edges) (Herman et al. 2000). There are a number of visualization techniques for representing graph structures. The size of the graph is one of the determining factors in the selection of the appropriate graph visualization. Comprehensibility is one of the major issues when one attempts to visualize a large graph as the display becomes clutter, and it becomes difficult to discern the difference between nodes and edges. The type of graph structure to be visualized, such as hierarchy, directed acyclic graph (DAG), and directed graph, also influences the graph visualization technique to be employed. For general graph, the most common approach is the **node-link diagram** which may utilize different layout algorithms, such as grid-variant algorithm (GVA), fast multi-scale (FMS), fast multipole multilevel (FM³), etc., for drawing nodes of a graph.

One particular type of graph structure is a tree, which is basically a hierarchical data structure. Tree structures are not limited to traditional organizational charts, file structures, subroutine call graphs, logic programming, and the like; they can also represent other complex data types, such as evolutionary and phylogenetic trees in biology or biochemical pathways and protein functions in chemistry. Traditional **tree layout** positions children nodes equally distant from parent ones, which reflects the intrinsic hierarchy of the data. A number of tree layout algorithms have been proposed, such as H-tree, balloon view, radial view, and others (see Herman et al. 2000 for more details). **Treemaps** (Johnson and Shneiderman 1991) is a particular method, based on the space-filling technique that maps hierarchical data into a rectangular-shaped 2D display, where 100% of the designated area is utilized. The area of each individual graph node will represent a numerical attribute of the data defined by the user (such as counts, size, etc.). The bounding boxes of each area represent a level in the hierarchy, and the location placement of items will recursively follow the hierarchical structure of the data.

Yet, as other visualization techniques, traditional graph visualization techniques fall short of supporting large graphs. Interactive approaches, such as **hyperbolic layout** (Lamping et al. 1995), which is a **focus+context** method that distorts the tree view using, for instance, fish-eye distortion, emerged in attempts to tackle the challenges of visualizing very large graphs. Graph clustering technique (Mukherjea et al. 1995) is another important data reduction technique for enabling the use of graph visualization for large graph data sets. However, data aggregation techniques are solely effective if each aggregated level can be effectively visualized as a node-link diagram and if the navigation between levels can maintain the necessary continuity so that users do not lose context (Elmqvist et al. 2008).

Adjacency matrix (Abello and van Ham 2004) has emerged as an alternative, effective technique for visualizing large networks. It couples a matrix representation with hierarchical aggregation to allow large network visualization and navigation. One of the greatest challenges of adjacency matrix is to determine the most appropriate ordering of rows and columns so that users can make sense of the graph data and extract insights. Elmqvist et al. (2008) propose a “zoomable” approach for adjacency matrix, named **Zoomable Adjacency Matrix Explorer**, which is a tool for large graph exploration at a scale of millions of nodes and edges. It allows users to “explore a graph at many levels, zooming and panning with interactive performance from an overview to the most detailed views” (Elmqvist et al. 2008, p. 215).

Nowadays, with the advent of the Internet, graphs, such as social network graphs, are in the order of billions of nodes, which pose unprecedented challenges to the visualization technique. For instance, Leskovec and Horvitz (2008) presented a study of 30 billion conversations generated by 240 million distinct Messenger accounts, which in turned created a communication graph of 180 million nodes and 1.3 billion undirected edges and a “buddy” graph of 9.1 billion edges. In order to gain insights from this massive graph, in addition to employing traditional social network analysis techniques, they utilized the concept

of **heat-maps** (or density) on top of traditional scatterplots and maps. In so doing, they displayed the density of the information for the various attributes of interest (such as number of communications, conversation duration, messages per conversation, and the like) as opposed to displaying individual notes and edges.

Data Aggregation Methods

Although the aforementioned visualization methods, namely, pixel-oriented, icon-based, and axis-based, attempt to tackle both multidimensionality and scalability issues, Yang et al. (2003) assert that they tend not scale well when data is wide (large numbers of dimensions) and tall (huge numbers of data points). In fact, they alone do not effectively scale even to moderately small data sets (Eick and Karr 2002). According to Liu et al. (2013), techniques that directly map each data point into a visual item tend not be well suited for visualizing very large data sets (tall data), due to occlusion and cluttering issues. Putting it simply, the challenge stems from how to “squeeze a billion records into a million pixels” (Shneiderman 2008). On a similar note, Elmqvist and Fekete (2010) assert that visualization techniques, such as scatterplots (Ward 2009), parallel coordinates (Inselberg and Dimsdale 1990), and treemaps (Johnson and Shneiderman 1991), alone do not handle large-scale data sets well, in that they result in clutter that hinders visual understanding. Hence, visualization techniques for big data exploration require the combination of visualization techniques, data processing, and interactive interfaces.

To address this problem, **data aggregation** techniques have been proposed. They aim to tackle both rendering and occlusion problems as they limit the number of items to be displayed (Elmqvist and Fekete 2010). For instance, Guo (2003) proposes a framework and approach for coordinating computational algorithms (e.g., feature selection) and interactive visualization to allow data analysts to explore high-dimensional data. According to him, feature selection is a nontrivial task that may render data visualization useless if not properly employed. He thus argues for a greater integration and coordination be-

tween data analytics and visualization techniques by means of a “human-centered and component-oriented knowledge discovery approach” (Guo 2003, p. 233) for big data exploration.

Data aggregation is part of a large set of data reduction techniques which also include **filtering** (e.g., Ahlberg and Shneiderman 1995), **sampling** (e.g., Bertini and Santucci 2006), and **clustering** (e.g., Jain et al. 1999). Both filtering and sampling techniques pin down a subset of the data that will be displayed by a standard visualization technique. The challenge is then to decide which subset to select. Different sampling techniques have been proposed, such as random sampling (every data point has the same probability of being selected), systematic sampling (a regular sampling interval/pattern is employed throughout the entire data set), and stratified sampling (disjoint subgroups are created and random and systematic sampling are applied in each group or “strata”) (Liu et al. 2013).

Data aggregation, on the other hand, does not reduce the data to be displayed but orderly group them out in “bins” – thus also known as **binned aggregation**. Bins can thus be defined as “adjacent intervals over a continuous range” (Liu et al. 2013, p. 423) or **density plots** (Shneiderman 2008). The benefit of this technique results from keeping both global patterns (e.g., density) and local features (e.g., outliers) in the same visualization while enabling multilevel resolution via different bin sizes. In sum, it focuses on limiting the “resolution” rather than the “data” to be displayed. On a separate note, Ellis and Dix (2007) systematically review and evaluate a number of data processing techniques relative to their effectiveness in addressing particular data clutter issues, including data aggregation. In particular, they look into their benefits and losses in clutter reduction techniques and, consequently, propose a guideline for future technique developments.

The ultimate goal of data aggregation methods is to improve visual scalability. Putting visualization and data aggregation techniques together sets the backstage against which visualization techniques will be able to effectively and efficiently interact with big data and consequently support data exploration. Choo and Park (2013)

nevertheless suggest that data analysis methods (or computational methods) need to be customized in order to effectively support visualization tasks. They argue that computational methods are often treated as “black boxes,” which hinder data analysts’ proper understanding of their internal “mechanics” in order to make appropriate adjustments and choices. On the other hand, they require intense computation and significant run-time, hindering real-time, interactive visualization. The authors thus propose customizing those methods based on the precision and convergence requirements of the visual exploration task at hand. Ultimately, the goal of data aggregation techniques is to provide data analysts with the most appropriate level of data resolution (or approximation) in order to allow them to interactively explore the data set and consequently extract deeper insights.

Interactive Techniques

Despite the recent advances in data analysis techniques, humans are still required to be involved so as to make sense of those large data sets. Current technologies fall short of general reasoning capabilities that would allow them to extract useful meanings out of those data sets, in particular when little is known about the data and the exploration goals are not fully defined. Thus current approaches for visual data exploration seek to integrate humans in the data exploration processes.

For visualization techniques to support (big) data exploration, they are required to allow data explorers to “slice and dice” large data sets by means of interactive visual interfaces. It should enable users to manipulate, search, and analyze those data sets in attempts to identify useful information and ultimately extract relevant insights to the problem at hand. In reality, users often start with no clear understanding or hypothesis about the data. Through the iterative process of data investigation, they incrementally construct meanings and a better comprehension of the data structures and contents, which allows them to start formulating an increasingly clearer hypothesis. The process of exploring large data sets is thus not simply a process of extracting information

from represented data patterns but a synergistic process of human-machine interaction by which machines contribute with their data processing and rendering power and humans contribute with their inherent reasoning and visual aptitudes.

Interactivity means not only adding humans to the loop but empowering them to effectively explore the data spaces (by quick testing hypotheses, switching views, and the like) at the rate of human thought (Heer and Kandel 2012). Visual exploration of big data thus requires an interplay between analytical reasoning interactive visual data manipulation and exploration. Visual data exploration workflow has been traditionally described by Shneiderman (2003) as “overview first, zoom/filter, details on demands” (a.k.a. the infamous *information seeking mantra*). However, big data demands a greater deal of data analyses (often automated) in order to allow more profound understandings of the data in question. In light of this, Keim et al. (2006) proposed a *visual analytics mantra*: “analyze first, show the important, zoom/filter, analyze further, details on demand.” This new *mantra* underscores the importance of combining and integrating data analytics methods and interactive visual interfaces. Putting it simply, it shows the role of employing **visual analytics** in support of big data exploration.

In the past 10 years, **visual analytics** emerged with the aim of further integrating those three critical aspects of big data exploration: visualization techniques, data analytics, and human visual reasoning (Keim et al. 2008b). This is a multidisciplinary field of research that combines analytical reasoning techniques, interactive visualization, and data analytics in order to tackle the challenges of handling massive, heterogeneous, and dynamic data sets. It enables the exploration of big data by providing technological tools and techniques by which users can manipulate, examine, select, and compare those complex data sets. To this end, it combines visual and data analysis methods, which allow progressive evaluation of the data and refinement of analysis results (Sun et al. 2013). By tapping into human’s inherent visual cognitive capabilities, visual analytics enables the discovery of critical visual similarities,

patterns, and relationships that allow incremental refinements of data models and consequently a deeper understanding of the data. In turn, data analytics transform the data so that it becomes better suited for its visualization. As a result, visual exploration aided by advanced data analytics provides an effective approach for exploring very large sets of data.

Future Directions of Research

Thomas and Cook (2006), back in 2006, proposed a list of issues to which the visualization research community was supposed to attend in order to address the data challenges they were starting to face. Six years later, Wong et al. (2012) outlined a list of ten major challenges in extreme-scale visual analytics – dealing with exabyte (10^{18}) data. Both also resonate with Heer and Kandel's (2012) questions for future research in interactive (big) data analysis. Interestingly, their lists and questions are still up-to-date and relevant today, with minor exceptions. In fact, the challenges have only abounded due to the data deluge of the past 20 years. Thus, it is reasonable to assert that the next advances in interactive visualization and visual analytics research will have to take into consideration the following challenges:

- To further expand visual analytic techniques to help data analysts to understand massive and continually growing collections of heterogeneous data. This implies, among other things, (1) being able to perform data and visual analyses with the data on memory (*in situ*) so as to allow human-thought speed (this is a major issue, in particular, when operating in cloud environments), (2) devising user-driven data reduction approaches so that data analysts would have more control of the process, and (3) devising new data summarization and triage for interactive queries.
- To deal with increasingly more complex and heterogeneous data. For instance, it will be important to devise frameworks and techniques

for spatial-temporal data analyses, semantic extraction of multimodal data, and multiple data-type fusion.

- To devise new approaches and techniques for tackling and understanding uncertain, incomplete, and often misleading data. In this context, it will be critical to create new mechanisms for representing evidence and uncertainty in extreme-scale data while minimizing visual “biases.”
- To support multiple levels of data abstraction and data-type integration in the same visual representation. And,
- To create new methods and approaches for allowing data analysts to more easily reason from extremely large data sets.

An open research field is to take a human-centric agenda to investigate the (social, physical, and virtual) environments for visual data exploration. As repeatedly pointed out here, human involvement is key in big data exploration. Hence, one future research direction would be to investigate how to improve human visual perception and understanding of big data. For one, alternative interaction techniques have been proposed, such as the use of augmented reality and virtual reality (Olshannikova et al. 2015), i.e., mixed reality as a means to allow data analysts to visually explore large data sets by embedding virtual content into the physical world in which they act. Reda et al. (2013) propose a hybrid reality environment, a large-scale visualization environment, in which the line between virtual environment and wall displays is blurred. They incorporate high-resolution, stereoscopic displays in order to provide a more immersive data exploration experience. On a different note, Heer et al. (2007) propose an approach for asynchronous collaboration on information visualization as a means of supporting collective sensemaking. Today's approaches and techniques tend to focus on individual data analysts exploring large, complex data sets by themselves. However, most of the data explorations are collective practices involving more than one individual. Thus,

creating conditions for supporting group visual data exploration becomes a necessary area of research.

While visualization techniques for big data exploration will continue to evolve with the development of incredibly richer visual metaphors, real-time interactive interfaces, sophisticated data analytic algorithms, ultrahigh-definition display walls, and the like, the most promising future technological directions will come out from an interdisciplinary collaboration between this field and artificial intelligence (AI). On the one hand, AI will become a site for employing advanced visual data exploration techniques in order to address major data/model understanding problems. On the other hand, visualization techniques will be able to benefit from the recent advances in AI, in particular in machine learning techniques (be they supervised or unsupervised) to allow the development of more sophisticated and complex data models, eliminating (at least in part) the need for human feature engineering. In this context, instead of direct data manipulation, data analysts will be interacting through the interface with models and algorithms. Today, they are already interacting with data models, but AI will turn that interaction more nuanced as models will increasingly become more like complex autonomous agents. This creates a host of new research questions not only relative to new visualization techniques for better conveying meaning out of machine learning-based data analyses but interactive interfaces through which data analysts will be able to meaningfully operate those increasingly more complex models.

Cross-References

- [Big Data Visualization Tools](#)
- [Visualization](#)

References

Abello J, van Ham F (2004) Matrix zoom: a visual interface to semi-external graphs. In: IEEE symposium on information visualization, pp 183–190. <https://doi.org/10.1109/INFVIS.2004.46>

- Ahlberg C, Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In: Baecker RM, Grudin J, Buxton WA, Greenberg S (eds) *Readings in human-computer interaction, interactive technologies*, 2nd edn, pp 450–456. Morgan Kaufmann (1995). <https://doi.org/10.1016/B978-0-08-051574-8.50047-9>. <https://www.sciencedirect.com/science/article/pii/B9780080515748500479>
- Andrews C, Endert A, Yost B, North C (2011) Information visualization on large, high-resolution displays: issues, challenges, and opportunities. *Inf Vis* 10(4):341–355. <https://colorado.idm.oclc.org/login?url=https://search.proquest.com/docview/1282260928?accountid=14503>. Copyright – SAGE Publications © Oct 2011; Document feature – Photographs; Tables; Illustrations; Last updated – 2014-12-11
- Becker RA, Cleveland WS (1987) Brushing scatterplots. *Technometrics* 29(2):127–142. <https://doi.org/10.1080/00401706.1987.10488204>
- Bertini E, Santucci G (2006) Give chance a chance: modeling density to enhance scatter plot quality through random data sampling. *Inf Vis* 5(2):95–110. <https://doi.org/10.1057/palgrave.ivs.9500122>
- Bieh-Zimmert O, Felden C (2014) Shaping unlimited patterns: a vision for state-of-the-art visual scalability. In: Proceedings of the 6th international conference on management of emergent digital EcoSystems (MEDES'14). ACM, New York, pp 13:72–13:77. <https://doi.org/10.1145/2668260.2668279>
- Bikakis N, Liagouris J, Krommyda M, Papastefanatos G, Sellis T: GraphVizdb: a scalable platform for interactive large graph visualization. In: 2016 IEEE 32nd international conference on data engineering (ICDE), pp 1342–1345 (2016). <https://doi.org/10.1109/ICDE.2016.7498340>
- Buja A, Cook D, Scientist DFSR (1996) Interactive high-dimensional data visualization. *J Comput Graph Stat* 5(1):78–99. <https://doi.org/10.1080/10618600.1996.10474696>
- Card SK, Mackinlay JD, Shneiderman B (1999) Readings in information visualization: using vision to think. Morgan Kaufmann, San Francisco
- Carr DB, Littlefield RJ, Nicholson WL, Littlefield JS (1987) Scatterplot matrix techniques for large N. *J Am Stat Assoc* 82(398):424–436. <https://doi.org/10.1080/01621459.1987.10478445>
- Chen C (2005) Top 10 unsolved information visualization problems. *IEEE Comput Graph Appl* 25(4):12–16. <https://doi.org/10.1109/MCG.2005.91>
- Choo J, Park H (2013) Customizing computational methods for visual analytics with big data. *IEEE Comput Graph Appl* 33(4):22–28. <https://doi.org/10.1109/MCG.2013.39>
- Cook KA, Thomas JJ (2005) Illuminating the path: the research and development agenda for visual analytics. Technical report, Pacific Northwest National Lab.(PNNL), Richland

- Eick SG, Karr AF (2002) Visual scalability. *J Comput Graph Stat* 11(1):22–43. <https://doi.org/10.1198/106186002317375604>
- Ellis G, Dix A (2007) A taxonomy of clutter reduction for information visualisation. *IEEE Trans Vis Comput Graph* 13(6):1216–1223. <https://doi.org/10.1109/TVCG.2007.70535>
- Elmqvist N, Fekete JD (2010) Hierarchical aggregation for information visualization: overview, techniques, and design guidelines. *IEEE Trans Vis Comput Graph* 16(3):439–454. <https://doi.org/10.1109/TVCG.2009.84>
- Elmqvist N, Do TN, Goodell H, Henry N, Fekete JD (2008) Zame: interactive large-scale graph visualization. In: 2008 IEEE Pacific visualization symposium, pp 215–222. <https://doi.org/10.1109/PACIFICVIS.2008.4475479>
- Guo D (2003) Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. *Inf Vis* 2(4):232–246. <https://doi.org/10.1057/palgrave.ivs.9500053>
- Heer J, Kandel S (2012) Interactive analysis of big data. *XRDS* 19(1):50–54. <https://doi.org/10.1145/2331042.2331058>
- Heer J, Viégas FB, Wattenberg M (2007) Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI'07). ACM, New York, pp 1029–1038. <https://doi.org/10.1145/1240624.1240781>
- Herman I, Melancon G, Marshall MS (2000) Graph visualization and navigation in information visualization: a survey. *IEEE Trans Vis Comput Graph* 6(1):24–43. <https://doi.org/10.1109/2945.841119>
- Inselberg A, Dimsdale B (1987) Parallel Coordinates for Visualizing Multi-Dimensional Geometry. In: Kunii TL (ed) Computer Graphics 1987. Springer, Tokyo
- Inselberg A, Dimsdale B (1990) Parallel coordinates: a tool for visualizing multi-dimensional geometry. In: Proceedings of the first IEEE conference on visualization: Visualization'90, pp 361–378. <https://doi.org/10.1109/VISUAL.1990.146402>
- Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. *ACM Comput Surv* 31(3):264–323. <https://doi.org/10.1145/331499.331504>
- Johnson B, Shneiderman B (1991) Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 2nd conference on visualization'91 (VIS'91). IEEE Computer Society Press, Los Alamitos, pp 284–291. <http://dl.acm.org/citation.cfm?id=949607.949654>
- Keim DA (2000) Designing pixel-oriented visualization techniques: theory and applications. *IEEE Trans Vis Comput Graph* 6(1):59–78. <https://doi.org/10.1109/2945.841121>
- Keim DA (2001) Visual exploration of large data sets. *Commun ACM* 44(8):38–44. <https://doi.org/10.1145/381641.381656>
- Keim DA, Mansmann F, Schneidewind J, Ziegler H (2006) Challenges in visual data analysis. In: Tenth international conference on information visualisation (IV'06), pp 9–16. <https://doi.org/10.1109/IV.2006.31>
- Keim DA, Mansmann F, Oelke D, Ziegler H (2008a) Visual analytics: combining automated discovery with interactive visualizations. In: Jean-Fran JF, Berthold MR, Horváth T (eds) Discovery science. Springer, Berlin/Heidelberg, pp 2–14
- Keim DA, Mansmann F, Schneidewind J, Thomas J, Ziegler H (2008b) Visual analytics: scope and challenges. In: Simoff SJ, Böhnen MH, Mazeika A (eds) Visual data mining: theory, techniques and tools for visual analytics. Springer, Berlin/Heidelberg, pp 76–90
- Lamping J, Rao R, Pirolli P (1995) A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI'95). ACM Press/Addison-Wesley Publishing Co., New York, pp 401–408. <https://doi.org/10.1145/223904.223956>
- Leskovec J, Horvitz E (2008) Planetary-scale views on a large instant-messaging network. In: Proceedings of the 17th international conference on world wide web (WWW'08). ACM, New York, pp 915–924. <https://doi.org/10.1145/1367497.1367620>
- Liu Z, Jiang B, Heer J (2013) Immens: real-time visual querying of big data. *Comput Graph Forum* 32(3pt4):421–430. <https://doi.org/10.1111/cgf.12129>
- Liu S, Maljovec D, Wang B, Bremer PT, Pascucci V (2015) Visualizing high-dimensional data: advances in the past decade. In: Proceeding of eurographics conference on visualization, pp 20151,115–127
- Meirelles I (2013) Design for information: an introduction to the histories, theories, and best practices behind effective information visualizations. Rockport Publishers. <https://books.google.com.br/books?id=RFb0AwAAQBAJ>
- Mukherjea S, Foley JD, Hudson S (1995) Visualizing complex hypermedia networks through multiple hierarchical views. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI'95). ACM Press/Addison-Wesley Publishing Co., New York, pp 331–337. <https://doi.org/10.1145/223904.223947>
- Nocke T, Schlechtweg S, Schumann H (2005) Icon-based visualization using mosaic metaphors. In: Ninth international conference on information visualisation (IV'05), pp 103–109. <https://doi.org/10.1109/IV.2005.58>
- North C, Shneiderman B (2000) Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In: Proceedings of the working conference on advanced visual interfaces (AVI'00). ACM, New York, pp 128–135. <https://doi.org/10.1145/345513.345282>
- Novotny M, Hauser H (2006) Outlier-preserving focus+context visualization in parallel coordinates. *IEEE Trans Vis Comput Graph* 12(5):893–900. <https://doi.org/10.1109/TVCG.2006.170>
- Olshannikova E, Ometov A, Koucheryavy Y, Olsson T (2015) Visualizing big data with augmented and virtual

- reality: challenges and research agenda. *J Big Data* 2(1):22. <https://doi.org/10.1186/s40537-015-0031-2>
- Reda K, Febretti A, Knoll A, Aurisano J, Leigh J, Johnson A, Papka ME, Hereld M (2013) Visualizing large, heterogeneous data in hybrid-reality environments. *IEEE Comput Graph Appl* 33(4):38–48. <https://doi.org/10.1109/MCG.2013.37>
- Shneiderman B (2003) The eyes have it: a task by data type taxonomy for information visualizations. In: Bederson BB, Shneiderman B (eds) *The craft of information visualization, interactive technologies*. Morgan Kaufmann, San Francisco, pp 364–371. <https://doi.org/10.1016/B978-155860915-0/50046-9>. <https://www.sciencedirect.com/science/article/pii/B9781558609150500469>
- Shneiderman B (2008) Extreme visualization: squeezing a billion records into a million pixels. In: *Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD'08)*. ACM, New York, pp 3–12. <https://doi.org/10.1145/1376616.1376618>
- Sun GD, Wu YC, Liang RH, Liu SX (2013) A survey of visual analytics techniques and applications: state-of-the-art research and future challenges. *J Comput Sci Tech* 28(5):852–867
- Thomas JJ, Cook KA (2006) A visual analytics agenda. *IEEE Comput Graph Appl* 26(1):10–13. <https://doi.org/10.1109/MCG.2006.5>
- Ward MO (2002) A taxonomy of glyph placement strategies for multidimensional data visualization. *Inf Vis* 1(3–4):194–210. <https://doi.org/10.1057/PALGRAVE.IVS.9500025>
- Ward MO (2008) Multivariate data glyphs: principles and practice. In: *Handbook of data visualization*. Springer, Berlin/Heidelberg, pp 179–198. https://doi.org/10.1007/978-3-540-33037-0_8
- Ward MO (2009) Linking and brushing. In: Liu L, Özsu MT (eds) *Encyclopedia of database systems*. Springer, Boston, pp 1623–1626
- Wong PC, Shen HW, Johnson CR, Chen C, Ross RB (2012) The top 10 challenges in extreme-scale visual analytics. *IEEE Comput Graph Appl* 32(4):63–67. <https://doi.org/10.1109/MCG.2012.87>
- Yang J, Ward MO, Rundensteiner EA, Huang S (2003) Visual hierarchical dimension reduction for exploration of high dimensional datasets. In: *Proceedings of the symposium on data visualisation 2003 (VISSYM'03)*. Eurographics Association, Aire-la-Ville, pp 19–28. <http://dl.acm.org/citation.cfm?id=769922.769924>

Visualizing Linked Data

Visualizing Semantic Data

Visualizing RDF Data

► Visualizing Semantic Data

Visualizing Semantic Data

Michael Luggen

eXascale Infolab, University of Fribourg,
Fribourg, Switzerland

Berne University of Applied Sciences, Bern,
Switzerland

Synonyms

Automated creation of infographics; Visualizing linked data; Visualizing RDF data

Definitions

Visualizing semantic data describes the task of using the additional self describing features of semantic data (Linked Data/RDF) to inform the process of creating vector or bitmap drawings.

Overview

Visualizations are representations of data solely created for humans. The primary objective of a visualization is to clearly and efficiently communicate information to a target audience. A visualization allows us to focus on particular aspects of data by choosing specific types of primitives and aggregates. Furthermore a visualization optimizes for efficiently delivering the intended message to the end user.

Creating meaningful and efficient visualizations of data is a difficult task. It integrates the understanding of the data at hand, the knowledge of valid aggregations and transformations, and finally the selection of the right type of visualization to convey the intended message. There exist multiple professions like infographic design, data

journalism, and data science specializing in these tasks. These professions rely on a variety of tools to create visualizations. The tools are informed by a plethora of implicit knowledge, based on raw data mostly.

With semantic data, i.e., data which is provided with additional information regarding its form and content, new possibilities open up for data visualization. The modeling of the data and its relations is defined by well-known vocabularies and ontologies. Furthermore, with the use of RDF (see <https://www.w3.org/RDF/>), semantic data can be represented in a common representation regardless of the form of its original or underlying data model.

Examples of such data models include:

- Dimensionally organized data (linear data models): time series, statistical data, and geographic data
- Structural data: trees, taxonomies, and graph data
- Annotated multimedia: images, audio, video, and 3D models

A tool without any external information and that is also missing knowledge on the data model at hand cannot propose any sensible visualization. Thus, there exist several tools for creating visualizations specific to a data model.

The process adopted to create a visual representation based on semantic data can be described end-to-end; this provides the inherent advantage that the outcome is reproducible by the system without further external intervention by the user or developer in case of corrections or updates to the underlying data.

Future research directions could propose systems which can fulfill the demand of efficient and effective visualizations by allowing users to craft visual representations based on their formulated intent combined with the inspection of the provided data.

Because semantic data in RDF is represented as graph data, a number of methods were proposed for visualizing this graph data as graph

drawings (node-link diagrams) (Graziosi et al. 2017). Recent examples on the web are VOWL (<http://vowl.visualdataweb.org/>) and LodLive (<http://en.lodlive.it/>). However, researchers have argued that there exist only a small number of specific use cases (e.g., exploration of ontologies Lohmann et al. 2016) where the graph representation is actually efficient for the task at hand. For most tasks, other forms of visualizations tend to be more efficient (Dadzie and Pietriga 2016).

In the following sections, the current state of research and available tools for the creation of visualizations with semantic data (in our example represented in RDF) are described. First, a number of *requirements regarding the data model* of the semantic data are introduced. Then, a *taxonomy for the overall task* of creating visualizations based on semantic data is provided. In addition, the different *steps necessary to create visualizations* based on semantic data are discussed. Finally, an overview on the *spectrum of tools* used for visualizations, from handcrafted to automated approaches, is provided.

Semantic Data Requirements

Semantic data provides information to machines which help to understand the data itself. Apart from having the data expressed in a valid form like RDF, it is necessary for the data to be modeled by an ontology or described in an RDF Schema. This allows the reuse of any visualization on similar datasets following the same ontology. In the following, a selection of ontologies which are leveraged for visualization techniques is described. The ontologies are organized by the data model they provide for visualization.

V

Dimensional

Dimensional data organizes related data points on a dimension; in the case of semantic data, it additionally describes its content. Typical examples of dimensional are the *time* dimension or dimensions from a data-specific categorical attribute like *gender* or *age*.

A good starting point for arbitrary dimensional data is the *Data Cube vocabulary* (<https://www.w3.org/TR/vocab-data-cube/>). Not only does this vocabulary allow for a clear definition of the dimensions and properties inside a dataset, it also defines additional descriptions like titles, dimension descriptions, and units which can be attached to the data points, dimensions, and datasets, respectively.

Another base for dimensional data is to use the standard mode output of *csv2rdf* (<https://www.w3.org/TR/csv2rdf/>). This is especially useful as the data to visualize is often provided in CSV format anyway.

Hierarchical/Structural

For the modeling of hierarchical and structural information, the use of the *SKOS Ontology* (<https://www.w3.org/TR/skos-reference/>) is widely spread. It is also feasible to represent structural information directly with the *OWL Web Ontology Language* (<https://www.w3.org/TR/owl-overview/>).

Geographical

Geo-spatially located data points often use the simple *Basic Geo Ontology* (<https://www.w3.org/2003/01/geo/>). For more complex geospatial data, one can refer to the *GeoSPARQL* (<http://www.opengis.net/standards/geosparql>) documentation which is using *well-known text (WKT)* (<http://www.geoapi.org/3.0/javadoc/org/opengis/referencing/doc-files/WKT.html>).

Miscellaneous

Further media data (pictures, video, or audio) can be described and annotated with the *Web Annotation Vocabulary* (<https://www.w3.org/TR/annotation-vocab/>). The power of semantic data also comes with the possibility to combine and mix different data models. This is possible as all the data models introduced above are organized by the same RDF formalisms.

Taxonomy

In this section, a high level taxonomy with the goal to situate the different available approaches

is provided. The taxonomy is influenced by prior work on visualization taxonomies: Brunetti et al. (2013) provide a formal framework in this context, while Graziosi et al. (2017) focus on the visualization of graph drawings.

The task of creating visualizations based on semantic data is split into the following steps:

1. View selection: when necessary, a specific subset of the available data is chosen.
2. Visual primitive selection: a fitting type of visualization is selected.
3. Data mapping: data then needs to be projected onto the visual primitive.

At the end of this section, interactive visualizations in line with this taxonomy are shortly discussed.

Selection of the Data View

The semantic data to be visualized can be accessed in multiple ways, from data dump files through server calls or a generic query language.

Typically, the data to be visualized is a part of all the available data points in a dataset. Hence, there is the need to select a view on the data, which is a subset of all available data. The decision on which data points are part of the visualization is determined by the application logic which also handles the interaction with the visualization itself.

The selection of the data view is dependent of primitives available for data access and filtering. In case the semantic data is preprocessed in a file, the selection was probably already done during its creation process. In case there is a (REST Style) API defined to provide the semantic data, the possible views are then defined through the API design.

On the contrary, if semantic data is consumed through a query language (like SPARQL), the application logic can define the view on the data in more detail. The generated visualizations also might have the potential to interact with the view selection mechanism in order to get updated iteratively.

Selection of the Visual Primitive

Once the decision is made about which data points should be visualized, selecting a suitable visual primitive is necessary. A visual primitive is a description of how to transform (render) the structured data on a canvas to create a visualization. Two approaches to select the appropriate visual primitives can be identified: an explicit selection during the data creation process or an implicit selection at run time.

Generally, the selection of the visual primitive is *explicitly* defined, either in the programming code itself or through annotating the data with hints which visualization to use.

For systems where the selection is *implicit*, the selection of the visual primitive becomes a ranking problem based on a catalog of available visual primitives. Rule-based systems leverage properties of the semantic data to decide on a visual primitive. Many properties can be leveraged in that sense, including the size or the dimension of data objects or the level of a hierarchy. In addition, the rule engine can also leverage features from the semantic model itself, for example, the use of specific ontologies (as introduced in the section above).

Rule-based engines can be further informed by the context of the application or use cases, including the mode of interaction with the user (role, language) and the current device(-type) she uses (desktop, mobile).

Future research is needed to provide applications to solve the selection of visual primitives with machine learning approaches based on the features described above.

Explicit and implicit approaches are not mutually exclusive, and systems profit from a combination of both these methods.

Data Mapping

Once the view on the data and the visual primitive has been fixed, both need to be connected.

Projection of the Data

For the majority of visual primitives (e.g., line and bar charts), the data needs to be projected from the graph representation of the semantic data onto one or multiple continuous dimensions.

If the data is consumed through SPARQL, this step is often solved early on by using a SELECT query. The SELECT query provides by design as an output a projection of the data to a table structure. This provides a dimensional structure which can be consumed by the mentioned visual primitives. The disadvantage of using a SELECT query is that the output of the query is no longer semantic data but rather a flat table. This has the drawback that additional annotations on the data that might be useful for the visualization are stripped away.

This is why several tools making an implicit visual selection use CONSTRUCT queries in SPARQL instead. The output of such a query is a subset of the source of the semantic data, but it keeps all semantic aspects.

Mapping Based on Semantic Data

If the decision is made to keep the semantic data intact all the way to the creation of the visual primitive, the need to bind the data from the graph structure to the input format of a visual primitive arises. Two approaches are possible in this context.

The more traditional approach is to leverage the information which is held in a TBOX (i.e., the data model, also called an ontology or a schema) to create the mapping for the visual primitive. Thus, the explicit conceptual model of a data structure which informs the mapping is kept. The advantage is that the ontology at hand defines the form of the content itself. This approach has the disadvantage that explicit formulation of the ontology needs to be available during the mapping process.

There are also frameworks which simply rely on the ABOX (the instance data itself). Here, the mapper inspects the data and the input expected by the visual primitive to discover a suitable mapping. In this case the expected input of the visual primitive figures as an implicit TBOX.

Interaction

The interaction with the visual primitives is generally orchestrated through the application logic. In addition, the different visual primitives often come with elements to interact with the

visualization itself. Possible interaction elements are filters on dimensions and values, controls for the selection of dimensions, or controls on the extent to which the data should be shown in the visual primitive.

These controls can either adapt the output of the visual primitive or trigger the demand for another visual primitive altogether.

Tools

In this section, a selection of the available research and tools related to the creation of visualizations based on semantic data is provided, with a focus on tools that create graphics rendered in web browsers. To set a common base for all tools, an overview of the facilities from a web browser to enable drawing of visual elements is first given.

In the first two sections below (*Visualization Libraries and Frameworks* and *SPARQL Specific Frameworks*), approaches where the visualization is explicitly defined are discussed. In the third section (*Selection and Mapping Frameworks*), the current state of research for utilizing implicit selection of visualization primitives is discussed.

Table 1 provides an overview of current tools and sets them in relation with the taxonomy introduced above.

The *HTML5 Canvas Element*, the *HTML WebGL Element*, and the *HTML SVG Element* are the most basic elements for creating visualizations. While the HTML5 Canvas is a pixel-based canvas which provides basic drawing facilities, WebGL provides a 3D context based on OpenGL to draw graphics, while SVG allows to create vector-based visualizations by dynamically creating SVG files. Web-based libraries and frameworks use one or multiple of these elements to perform the visualization inside a web browser.

Visualization Libraries and Frameworks

Visualization libraries differ greatly in their flexibility. A plethora of libraries (<https://developers.google.com/chart/>, <http://www.chartjs.org/>, <https://www.highcharts.com/>) provide catalogs of visual primitives, which expect the input data

to be in a specific format. Most of these libraries provide only basic visualization possibilities.

The most prominent example of a more flexible framework is *D3.js* (<http://d3js.org>). *D3.js* provides developers with a set of tools to create visual primitives that are highly customizable.

SPARQL Specific Frameworks

Sgvizler Skjaeveland (2012) is the first widely used framework to create visualizations directly based on semantic data. Sgvizler fetches data from a SPARQL endpoint with SELECT queries and relies heavily on Google Charts for the visual primitives. It also provides an interface to write custom plug-ins which to create arbitrary visualizations.

The *D3SPARQL* project Katayama (2014) is comparable to Sgvizler in its functionality, as it provides a set of visual primitives based on *D3.js*.

Finally, the D3.js component *d3-sparql* (<https://github.com/zazuko/d3-sparql>) seamlessly integrates in the D3.js code base. It is a drop-in replacement for the d3-csv component. In that way, existing D3.js visualization can be informed through data originating from a SPARQL endpoint.

Selection and Mapping Frameworks

The frameworks presented in the following all provide an implicit selection of the visual primitive.

Tabulator Berners-Lee et al. (2006) introduced a generic data browser, along with the concept of an implicit selection of a user interface component informed through semantic data. Some of these components were featuring visual primitives.

Balloon synopsis (Schlegel et al. 2014) is a jQuery Plugin which allows to easily enhance a website with semantic data. For the mapping *ontology templates* were introduced.

In Uduvudu (Luggen et al. 2015) a framework for adaptive user interface generation is proposed. The Uduvudu framework uses a two-step ruled-based approach to select the user interface elements and the visual primitives. By separating the selection and the rendering step, it puts a focus on the reusability by maximizing dynamic

Visualizing Semantic Data, Table 1 Presented tools located in the introduced taxonomy

Tool/Framework	View selection	Visual primitive	Mapping
Sgvizler	SELECT	Explicit	Hard coded catalog
D3SPARQL	SELECT	Explicit	Hard coded catalog
d3-sparql	SELECT	n/a	n/a
Tabulator	None (Browser)	Implicit/rule-based	In code
Balloon synopsis	Ontology template	Implicit/rule-based	Handlebars
Uduvudu	CONSTRUCT	Implicit/rule-based	Matchers
RSLT	SQM/DQM	Implicit/rule-based	HTML templates
Linked data reactor	Scopes	Implicit/rule-based	Web component
Phuzzy	None (Browser)	Implicit/rule-based	In code (plugin)

composition capabilities. Furthermore, the framework can take the context of the end user (user preferences, used device, language) into account whenever available.

RSLT (Peroni and Vitali 2015) provides a library inspired by the XSLT transformation language. The semantic data is transformed from its RDF model by using templates. The templates are matched with a newly proposed selector language.

Linked Data Reactor (Khalili 2016) is a framework to build reactive applications realized with hierarchical *web components*. The mapping to a component can be done on the level of a dataset, resource, property, or value.

Phuzzy (Regalia et al. 2017) introduces the capability of dynamically downloading plug-ins providing rendering facilities for visual primitives.

Cross-References

- ▶ [Big Data Visualization Tools](#)
- ▶ [Dimension Reduction](#)
- ▶ [Graph Data Models](#)
- ▶ [Graph Query Languages](#)
- ▶ [Linked Geospatial Data](#)
- ▶ [Schema Mapping](#)
- ▶ [Visual Graph Querying](#)
- ▶ [Visualization Techniques](#)

References

Berners-Lee T, Chen Y, Chilton L, Connolly D, Dhanaraj R, Hollenbach J, Lerer A, Sheets D (2006) Tabulator:

exploring and analyzing linked data on the semantic web. In: Proceedings of the 3rd international semantic web user interaction workshop, Athens, Citeseer, vol 2006, p 159

Brunetti JM, Auer S, Garcia R, Klimek J, Necasky M (2013) Formal linked data visualization model. In: Proceedings of international conference on information integration and web-based applications & services, IIWAS'13. ACM, New York, pp 309:309–309:318. <https://doi.org/10.1145/2539150.2539162>

Dadzie AS, Pietriga E (2016) Visualisation of linked data – reprise | www.semantic-web-journal.net, <http://www.semantic-web-journal.net/content/visualisation-linked-data-%E2%80%93-reprise>

Graziosi A, Iorio AD, Poggi F, Peroni S (2017) Customised visualisations of linked open data. In: Ivanova V, Lambrix P, Lohmann S, Pesquita C (eds) Proceedings of the third international workshop on visualisation and interaction for ontologies and linked data co-located with the 16th international semantic web conference (ISWC 2017), Vienna, 22 Oct 2017, CEUR-WS.org, CEUR workshop proceedings, vol 1947, pp 20–33. <http://ceur-ws.org/Vol-1947/paper03.pdf>

Katayama T (2014) D3sparql: JavaScript library for visualization of SPARQL results. In: SWAT4LS, Berlin, 9–11 Dec 2014. Citeseer

Khalili A (2016) Linked data reactor: a framework for building reactive linked data applications. In: Troncy R, Verborgh R, Nixon LJB, Kurz T, Schlegel K, Sande MV (eds) Joint proceedings of the 4th international workshop on linked media and the 3rd developers hackshop co-located with the 13th extended semantic web conference ESWC 2016, Heraklion, 30 May 2016, CEUR-WS.org, CEUR workshop proceedings, vol 1615. <http://ceur-ws.org/Vol-1615/semdevPaper4.pdf>

Lohmann S, Negru S, Haag F, Ertl T (2016) Visualizing ontologies with VOWL. Semantic Web 7(4):399–419. <https://doi.org/10.3233/SW-150200>, <https://content.iospress.com/articles/semantic-web/sw200>

Luggen M, Gschwend A, Anrig B, Cudré-Mauroux P (2015) Uduvudu: a graph-aware and adaptive UI engine for linked Data. In: Bizer C, Auer S, Berners-Lee T, Heath T (eds) Proceedings of the workshop on linked data on the web, LDOW 2015, co-located with the

- 24th international world wide web conference (WWW 2015), Florence, 19th May 2015, CEUR-WS.org, CEUR workshop proceedings, vol 1409. <http://ceur-ws.org/Vol-1409/paper-07.pdf>
- Peroni S, Vitali F (2015) RSLT: R. D. F. stylesheet language transformations. In: Verborgh R, Sande MV (eds) Proceedings of the ESWC developers workshop 2015 co-located with the 12th extended semantic web conference (ESWC 2015), Portorož, 31 May 2015, CEUR-WS.org, CEUR workshop proceedings, vol 1361, pp 7–13. <http://ceur-ws.org/Vol-1361/paper2.pdf>
- Regalia B, Janowicz K, Mai G (2017) Phuzzy.link: a SPARQL-powered client-sided extensible semantic web browser. In: Proceedings of the third international workshop on visualization and interaction for ontologies and linked data (VOILA'17), Vienna, CEUR-WS.org, CEUR workshop proceedings, vol 1947
- Schlegel K, Weißgerber T, Stegmaier F, Granitzer M, Kosch H (2014) Balloon synopsis: a jQuery plugin to easily integrate the semantic web in a website. In: Proceedings of the ISWC developers workshop 2014, Riva del Garda, 19–23 Oct 2014
- Skjaeveland MG (2012) Sgvizler: a javascript wrapper for easy visualization of SPARQL result sets. In: Extended semantic web conference. Springer, Berlin/Heidelberg, pp 361–365

W

Weaker Consistency Models/Eventual Consistency

Faisal Nawab
University of California, Santa Cruz, CA, USA

Definitions

Weaker consistency and eventual consistency models are classes of consistency in data management systems that trade off consistency for performance.

Overview

Data management systems have employed replication and distribution of data for many objectives, including fault tolerance, load balancing, and read availability, among others (Kemme et al. 2010; Bernstein and Goodman 1981). Having data be distributed or replicated on various nodes raises a problem of how to coordinate access to data across all nodes. Specifically, a client accessing data may affect the state of multiple nodes concurrently. With simultaneous access from different clients, the distributed state of the data may be inconsistent due to overwrites, race conditions, and other concurrency anomalies. Distributed and replicated data management

systems are more susceptible to such problems compared to parallel data management systems because the communication delay between nodes is orders of magnitude larger than inter-process communication.

In environments where the communication latency is large, such as geo-scale distributed or replicated data management systems, the coordination cost is exacerbated even more. Additionally, environments with a large number of machines also suffer from high coordination cost. These two characteristics, geo-scale and large-scale deployments, are increasingly adopted due to their benefits, such as increasing the levels of fault tolerance and availability. Also, they are increasingly adopted due to their accessibility through public cloud providers that offer a large amount of resources at various data centers around the world. This has made the coordination cost a pressing problem for data management systems. One of the main approaches that enjoyed wide success in tackling coordination cost is the relaxation and weakening of consistency semantics. Unlike strong consistency semantics, weaker and relaxed semantics can potentially lead to anomalies and thus may be unsuitable for some classes of applications. The explorations in weaker and relaxed consistency semantics aim at understanding the anomalies and models of weaker and relaxed consistency in addition to building data management protocols that achieve these consistency levels while maximizing performance.

Key Research Findings

This section surveys the key advancements in weak and relaxed consistency semantics, with an emphasis on cloud and geo-scale deployments.

Eventual Consistency

Eventual consistency trades off strong consistency for better performance. It has been adopted widely in environments where the coordination cost and performance demands are high. A major influence to this line of work is due to the CAP theorem (Gilbert and Lynch 2002; Brewer 2000). CAP stands for three properties: consistency (different copies or multisite copies of data are consistent), availability (ability to always respond to requests), and partition tolerance (a network failure that separates nodes does not interrupt operation). The conjecture is that only two out of the three properties can be achieved in a replicated system. The implication of this conjecture is that systems that are available and partition-tolerant cannot be consistent. This has influenced eventual consistency, which focuses on availability and partition tolerance in the expense of consistency.

Eventual consistency, broadly defined, is the guarantee that in the absence of updates, all copies of data converge to a common state eventually. Therefore, the results returned at a given time might be inconsistent with a latter view of the data. Although this is a weak model of consistency, it enables higher levels of performance and is especially favorable for applications that require high performance and availability. Methods that resemble eventual consistency in the context of strongly consistent systems dates back to Thomas' majority voting algorithm (Thomas 1979). In that work, multiple copies of a database may have different versions of a data item as a number of write operations may be applied in a different orders in different copies. To choose between the write operations, a database copy picks the one with the largest time stamp. Because all copies pick the write with the largest timestamp, eventually all copies agree on the value of the data item in the absence of further write operations to that data item. In general,

this convergence property relies on the ability to propagate across nodes and the ability to reach a consistent ordering. Often, eventual consistency has been used as a guarantee for replication services that are used by higher-level services that guarantee more complex, stronger consistency guarantees (Terry et al. 1994).

However, more recently, as the need for available high-performance systems grew, eventual consistency has been used as the consistency guarantee observed by programmers (DeCandia et al. 2007; Cooper et al. 2008; Bailis and Ghodsi 2013; Lakshman and Malik 2010). The wide adoption of eventual consistency has shown that a weak model of consistency is capable of supporting large services and a wide range of applications. More importantly, it demonstrated how an eventually consistent system can be built for real-world scenarios by providing a better understanding of the guarantee of eventual consistency, the design principles for building eventually consistent systems, and the best practices in programming with eventual consistency.

An important characteristic of eventually consistent system is that for many workloads, the observed state of data resembles that of a strongly consistent system most of the time (Bailis et al. 2012). Probabilistically bounded staleness (PBS) (Bailis et al. 2012) is an example of modeling and quantifying this behavior on Cassandra, a popular eventually consistent system (Lakshman and Malik 2010). PBS models the behavior and the expected consistency of read data by estimating the probability of deviating from strong consistency. Specifically, given a certain configuration and workload, it is possible to estimate the probability of reading the most up-to-date data value (representing the consistent state). This, and similar models, can also be used to configure the eventually consistent system to satisfy probabilistic guarantees. This is significant as many systems allow configuring the parameters that affects the probability of a consistent outcome.

An enabler of eventual consistency is the set of practical designs and principles that were used to build eventually consistent systems. Although the

basis of many eventually consistent systems is the guarantee of eventual convergence, many higher-level access methods were built on top of that basis to provide better access semantics while maintaining the performance and availability features of eventual consistency. PNUTS (Cooper et al. 2008) is an example of such systems. In PNUTS, data is replicated in many copies, and one of these copies is a primary. A primary receives all write requests and propagates them to secondary copies. PNUTS leverages this write pattern to order writes at the primary. This results in a useful, simple consistency model – called per-record timeline consistency – where writes, which are ordered at the master, are guaranteed to be ordered at the secondary copies. Another example from PNUTS of a higher-level access method on top of eventually consistent data systems is that it provides three ways to read data. A **read-latest** forces the read to be served from the primary, a **read-critical** forces the read to go to a copy that has a recent enough copy according to an input timestamp, and a **read-any** can be sent to any copy and thus provides the highest performance.

In addition to the ability to provide stronger notions of consistency on top of eventual consistency, providing alternative methods to perform the same operation allows the developer to control the consistency-performance trade-off. Some systems take this concept further and build a layer on top of an eventually consistent system to provide stronger consistency guarantees (Zhang et al. 2015; Bailis et al. 2013). TAPIR (Zhang et al. 2015) provides strongly consistent replication on top of an eventually consistent replication service, and Bolt-on (Bailis et al. 2013) provides causally consistent replication on top of an eventually consistent data store.

Another facilitator of the success of eventually consistent systems is the way programmers use them. Being aware of the underlying consistency level may influence programmers to adopt data structures and programming techniques that ameliorate the negative outcomes of eventual consistency. One such class of data structures is the class of commutative operations. Commutative

operations are ones that can be reordered arbitrarily while leading to the same final state, such as increment/decrement operations. Convergent and commutative replicated data types (CRDTs) (Shapiro et al. 2011) is an approach to programming that ensures that the execution of a set of operations would be guaranteed to converge with the final outcome of the execution of the same operations at any order. There have been many data structures based on CRDTs; however, they are constrained and are not applicable to all operation types. Another approach adopted in eventually consistent systems is the ordering of execution at all copies, which guarantees convergence to the same value, such as the example of PNUTS' per-record timeline consistency (Cooper et al. 2008) and the example of Thomas' write rule (Thomas 1979). In PNUTS, a centralized authority orders writes, and in Thomas' write rule, unique time stamps guarantee order. Ordering writes could also be relaxed, adopting partial ordering techniques such as vector clocks (Mattern et al. 1989). The benefit of partial ordering in this context is that ordering is not enforced for concurrent writes, which may reduce overhead. Although there exist a lot of techniques and approaches to reduce the negative outcomes of eventual consistency, they are not applicable to many operations and tasks. For applications that use such operations with eventual consistency, conflict resolution is tricky, requiring intervention or specialized conflict-resolution logic. Helland and Campbell (2009) propose a framework to reason about such conflict resolution methods. Specifically, they divide application logic fall into *memories* which are the states of a local node, *guesses* which are the actions that a node takes based on its current (eventually consistent) knowledge, and *apologies* which are the reactions to mistaken guesses that led to a conflict. An apology can be through human intervention but can also be conducted through carefully crafted conflict-resolution modules.

W

Relaxed Consistency

A large body of work explored the spectrum of the consistency-performance trade-off, that is,

works that explore consistency levels that are stronger than eventual consistency but weaker than strong notions of consistency such as serializability (Bernstein et al. 1987) and linearizability (Herlihy and Wing 1990). The motivation of these explorations is that strong consistency and eventual consistency are two extremes in the spectrum and that for many applications, the consistency and performance requirements are diverse. In such applications, an intermediate consistency level may be ideal as it provides the necessary consistency guarantees while maximizing performance by reducing the overhead of coordination.

Causal consistency (Lamport 1978) is an example of relaxed consistency levels. It is a guarantee to order events according to their causal relations. Causal relations are defined to be one of the following: (1) It has two events that are generated in the same node are totally ordered. That is, they maintain the same order at all nodes. (2) It has an event that is received by a node is ordered after all previous events at that node. This is called the happens-before relation. (3) The causal order is transitive, and thus a chain of causal relations translates to be a causal relation. An alternate definition of causality may depends on the type of operations. For read and write operations, for example, a happens-before relation is substituted with a gets-from relation from a write operation to a read operation if the read has read the value of the write operation (Ahmad et al. 1990; Lloyd et al. 2011). This definition of causal consistency is widely used in data management systems as it is specialized for read and write operations on data items.

An interesting feature of causal consistency is that it does not require synchronous coordination across nodes. An operation that is executed in a node may be served without any coordination with other nodes (propagation of updates and events can happen asynchronously). This makes it possible for causally consistent systems to be both available and partition-tolerant while providing a notion of consistency stronger than eventual consistency. This motivated an exploration of causal consistency in the context of cloud and

geo-scale deployments with high coordination cost (Lloyd et al. 2011, 2013; Bailis et al. 2013; Nawab et al. 2015; Du et al. 2013a). However, an issue that may arise with causal consistency is that multiple write operations on the same data object might not converge to the same value at all locations. This is because two write operations that are issued at different nodes may have no causal relation between each other. This has been tackled by some causally consistent systems. An example is the causally consistent geo-scale data store COPS (Lloyd et al. 2011), where data objects are guaranteed to converge in all locations. COPS formally defines the convergence property in the context of causality by introducing the causal+ consistency. Causal+ consistency adds convergence as a guarantee to traditional causal consistency.

Another challenge with causal consistency is reasoning about causally consistent transactions. Causal relations are traditionally defined in the level of operations and events. It is not immediately clear how these relations can be applied to data transactions that consist of bundles of read and write operations. Raynal et al. (1996) tackle this problem by defining causal-based guarantees for transactional access: causal consistency and causal serializability. Causal consistency defines a causal relation between two transactions if one transaction contains a read operation that reads a value written by the other transaction. Causal serializability augments causal consistency with a guarantee that all writes are observed with a sequential order in all nodes. This addition of a sequential order makes it necessary to synchronously coordinate with other nodes for write operations. However, the other types of access (causal consistency transactions and causal serializability read transactions) can be performed without synchronous coordination. More recently, COPS (Lloyd et al. 2011) and Eiger (Lloyd et al. 2013) explore adding transactional semantics to causal consistency. COPS introduces a read-only transaction interface that is causally consistent. COPS guarantees the consistency of the read-only transaction via a two-phase protocol that,

in the first phase, reads a local snapshot of the data and, in the other phase, updates the read data items with any missing dependencies. Eiger explores similar mechanisms to support both read-only and write-only causally consistent transactions.

Snapshot isolation (SI) is another relaxed consistency guarantee that is widely used in database management systems (Berenson et al. 1995). SI was intended to reduce the concurrency control overhead of serializability. It is defined as a guarantee to (1) read a consistent snapshot of the data and (2) avoid write-write conflicts between transactions. This relaxed notion of consistency, compared to serializability, introduces a short fork anomaly. In this anomaly, two concurrent transactions that write over each other's read data can both commit if their write sets are disjoint. For example, consider two transactions, t_1 and t_2 , that both read the same versions of x and y . Transaction t_1 writes a new value of x , and t_2 writes a new value of y . Both transactions can commit with SI but not serializability. An anomaly is caused when a future transaction reads the values of x and y which do not reflect a serial ordering of transactions. Despite this anomaly, SI has shown to be useful for many applications and to enable higher performance. However, because write conflicts must still be detected, SI requires synchronous coordination between nodes.

SI has been explored in the context of cloud and geo-scale deployments to reduce the expensive cost of coordination (Sovran et al. 2011; Daudjee and Salem 2006; Lin et al. 2005, 2007; Du et al. 2013b). In these bodies of work, efficient designs of SI are explored. Additionally, SI is relaxed further to accommodate geo-scale deployments and enable higher performance in a new consistency level called parallel snapshot isolation (PSI) (Sovran et al. 2011). PSI relaxes SI in a number of ways. PSI requires that a transaction reads a snapshot of the local copy in the data center rather than a snapshot of all copies in all data centers. Also, it extends the definition of write-write conflict to accommodate for the multi-data center nature of geo-scale de-

ployments. Finally, it introduces a causal relation between transactions at different data centers; where a transaction that commits before another starts, then they must be ordered. The relaxation of the snapshot guarantee and extension of write-write conflict definition enables PSI systems to replicate asynchronously between data centers in the absence of write-write conflicts. However, PSI introduces a long-fork anomaly that is similar to the short fork anomaly, but with the possibility of intermediate steps (transactions) before the anomaly is observed (Sovran et al. 2011).

Additionally, there have been other relaxed consistency notions with various consistency and performance attributes, many of which have been explored in the context of cloud and geo-scale deployments. One such consistency guarantee is read committed that is the default consistency level in MDCC, a Paxos-based protocol (Kraska et al. 2013; Pang et al. 2014). Read committed guarantees that only committed values are read. Read committed introduces a lost update anomaly, which is an anomaly that arises when a transaction overwrites the writes of another transaction, rendering them lost. MDCC augments read committed with a guarantee to detect write-write conflicts, which removes the possibility of lost update anomalies. MDCC also incorporates various relaxed notions of consistency such as atomic visibility that ensures that all or none of a transaction's effects are observed and non-monotonic snapshot isolation (Ardekani et al. 2013) which relaxes SI in a way similar to PSI.

An alternative way to provide consistency guarantees is from the context of a client's session, which is what a single client does and observes (Terry et al. 1994). These guarantees include *read your writes* which guarantees that a read observes all the client's writes, *monotonic reads* that guarantees that reads reflect a progressive state of data, *writes follow reads* that guarantee that writes are observed after the reads they depend on, and *monotonic writes* that guarantee an ordered propagation of writes. These guarantees can be selectively combined and implemented on top of an

eventually consistent data store, which allows customizing the solution to the needs of specific applications.

Conclusion

The high cost of coordination, which is exacerbated in cloud and geo-scale environments, motivates the exploration of weaker and relaxed forms of consistency. These forms of consistency enable higher performance while providing some weaker forms of consistency guarantees. The spectrum of the consistency-performance trade-offs includes many protocols that have various performance and consistency characteristics. The study and exploration of points in this spectrum help identify the suitable consistency levels for various classes of applications. Also, an important problem is finding efficient designs of relaxed and weak consistency notions to maximize the performance gains of relaxing consistency. The adoption of weak and relaxed consistency models has also impacted how programmers interact with storage systems by navigating the best practices of using weaker and relaxed guarantees in addition to reasoning about the consistency of applications.

Examples of Application

Weaker and relaxed consistency models are used by various data storage engines for the web; traditional database management systems, in the cloud; and geo-scale deployments.

Future Directions for Research

The growth and evolution of computing platforms (e.g., geo-scale deployments and hardware-based techniques) and application classes (e.g., data science and machine learning) invite a continued exploration of consistency guarantees in the consistency-performance trade-off to better suit these platforms and applications.

Cross-References

- ▶ Coordination Avoidance
- ▶ Geo-Replication Models
- ▶ Geo-Scale Transaction Processing

References

- Ahamad M, Neiger G, Kohli P, Burns J, Hutto P, Anderson T (1990) Causal memory: definitions, implementation and programming. *IEEE Trans Parallel Distrib Syst* 1:6–16
- Ardekani MS, Sutra P, Preguiça N, Shapiro M (2013) Non-monotonic snapshot isolation. arXiv preprint arXiv:13063906
- Bailis P, Ghodsi A (2013) Eventual consistency today: limitations, extensions, and beyond. *Queue* 11(3):20:20–20:32. <http://doi.acm.org/10.1145/2460276.2462076>
- Bailis P, Venkataraman S, Franklin MJ, Hellerstein JM, Stoica I (2012) Probabilistically bounded staleness for practical partial quorums. *Proc VLDB Endow* 5(8):776–787
- Bailis P, Ghodsi A, Hellerstein JM, Stoica I (2013) Bolt-on causal consistency. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD’13. ACM, New York, pp 761–772. <http://doi.acm.org/10.1145/2463676.2465279>
- Berenson H, Bernstein P, Gray J, Melton J, O’Neil E, O’Neil P (1995) A critique of ansi SQL isolation levels. pp 1–10. <http://doi.acm.org/10.1145/223784.223785>
- Bernstein PA, Goodman N (1981) Concurrency control in distributed database systems. *ACM Comput Surv (CSUR)* 13(2):185–221
- Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Reading
- Brewer EA (2000) Towards robust distributed systems (abstract). In: Proceedings of the nineteenth annual ACM symposium on principles of distributed computing, PODC’00. ACM, New York, pp 7–. <http://doi.acm.org/10.1145/343477.343502>
- Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Puz N, Weaver D, Yerneni R (2008) Pnuts: Yahoo!’s hosted data serving platform. *Proc VLDB Endow* 1(2):1277–1288. <https://doi.org/10.14778/1454159.1454167>
- Daudjee K, Salem K (2006) Lazy database replication with snapshot isolation. In: Proceedings of the 32nd international conference on very large data bases, VLDB Endowment, VLDB’06, pp 715–726. <http://dl.acm.org/citation.cfm?id=1182635.1164189>
- DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon’s highly available key-value store. In: Proceedings of twenty-first ACM

- SIGOPS symposium on operating systems principles, SOSP'07. ACM, New York, pp 205–220. <http://doi.acm.org/10.1145/1294261.1294281>
- Du J, Elnikety S, Roy A, Zwaenepoel W (2013a) Orbe: scalable causal consistency using dependency matrices and physical clocks. In: Proceedings of the 4th annual symposium on cloud computing, SOCC'13. ACM, New York, pp 11:1–11:14. <http://doi.acm.org/10.1145/2523616.2523628>
- Du J, Elnikety S, Zwaenepoel W (2013b) Clock-si: snapshot isolation for partitioned data stores using loosely synchronized clocks. In: Proceedings of the 2013 IEEE 32nd international symposium on reliable distributed systems, SRDS'13. IEEE Computer Society, Washington, pp 173–184. <https://doi.org/10.1109/SRDS.2013.26>
- Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59. <http://doi.acm.org/10.1145/564585.564601>
- Helland P, Campbell D (2009) Building on quicksand. arXiv preprint arXiv:09091788
- Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492. <http://doi.acm.org/10.1145/78969.78972>
- Kemme B, Jimenez-Peris R, Patino-Martinez M (2010) Database replication. Synth Lect Data Manage 2(1):1–153. <http://www.morganclaypool.com/doi/abs/10.2200/S00296ED1V01Y201008DTM007>
- Kraska T, Pang G, Franklin MJ, Madden S, Fekete A (2013) Mdcc: multi-data center consistency. In: Proceedings of the 8th ACM European conference on computer systems, EuroSys'13. ACM, New York, pp 113–126. <http://doi.acm.org/10.1145/2465351.2465363>
- Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. SIGOPS Oper Syst Rev 44(2):35–40. <http://doi.acm.org/10.1145/1773912.1773922>
- Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565. <http://doi.acm.org/10.1145/359545.359563>
- Lin Y, Kemme B, Patiño Martínez M, Jiménez-Peris R (2005) Middleware based data replication providing snapshot isolation. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD'05. ACM, New York, pp 419–430. <http://doi.acm.org/10.1145/1066157.1066205>
- Lin Y, Kemme B, Patino-Martinez M, Jimenez-Peris R (2007) Enhancing edge computing with database replication. In: Proceedings of the 26th IEEE international symposium on reliable distributed systems, SRDS'07. IEEE Computer Society, Washington, pp 45–54. <http://dl.acm.org/citation.cfm?id=1308172.1308219>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 401–416. <http://doi.acm.org/10.1145/2043556.2043593>
- Lloyd W, Freedman MJ, Kaminsky M, Andersen DG (2013) Stronger semantics for low-latency geo-replicated storage. In: Proceedings of the 10th USENIX conference on networked systems design and implementation, NSDI'13. USENIX Association, Berkeley, pp 313–328. <http://dl.acm.org/citation.cfm?id=2482626.2482657>
- Mattern F et al (1989) Virtual time and global states of distributed systems. Parallel Distrib Algoritm 1(23): 215–226
- Nawab F, Arora V, Agrawal D, El Abbadi A (2015) Charrots: a scalable shared log for data management in multi-datacenter cloud environments. In: Proceedings of the 18th international conference on extending database technology, EDBT 2015, Brussels, pp 13–24, 23–27 Mar 2015. <https://doi.org/10.5441/002/edbt.2015.03>
- Pang G, Kraska T, Franklin MJ, Fekete A (2014) Planet: making progress with commit processing in unpredictable environments. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD'14. ACM, New York, pp 3–14. <http://doi.acm.org/10.1145/2588555.2588558>
- Raynal M, Thia-Kime G, Ahamad M (1996) From serializable to causal transactions. In: Proceedings of the fifteenth annual ACM symposium on principles of distributed computing. ACM, p 310
- Shapiro M, Preguiça N, Baquero C, Zawirski M (2011) Convergent and commutative replicated data types. Bull Eur Assoc Theor Comput Sci (104):67–88
- Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: Proceedings of the twenty-third ACM symposium on operating systems principles, SOSP'11. ACM, New York, pp 385–400. <http://doi.acm.org/10.1145/2043556.2043592>
- Terry DB, Demers AJ, Petersen K, Spreitzer MJ, Theimer MM, Welch BB (1994) Session guarantees for weakly consistent replicated data. In: Proceedings of the third international conference on parallel and distributed information systems. IEEE, pp 140–149
- Thomas RH (1979) A majority consensus approach to concurrency control for multiple copy databases. ACM Trans Database Syst (TODS) 4(2):180–209
- Zhang I, Sharma NK, Szekeres A, Krishnamurthy A, Ports DRK (2015) Building consistent transactions with inconsistent replication. In: Proceedings of the 25th symposium on operating systems principles, SOSP'15. ACM, New York, pp 263–278. <http://doi.acm.org/10.1145/2815400.2815404>

W

Web of Data

► Linked Data Management

(Web/Social) Graph Compression

Paolo Boldi¹ and Sebastiano Vigna²

¹Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

²Università degli Studi di Milano, Milano, Italy

Synonyms

Compressed representations for complex networks; Data structures for graphs; Graph compression

Definitions

In this section, we discuss the problem of representing large graphs in core memory using suitable compressed data structures; after defining the problem, we survey the most important techniques developed in the last decade to solve it, highlighting the involved trade-offs.

A *graph* is a pair (N, A) , where N is the set of *nodes* and $A \subseteq N \times N$ is the set of *arcs*, that is, a directed adjacency relation. We use n for the number of nodes, that is, $n = |V|$, and write $x \rightarrow y$ when $(x, y) \in A$.

Overview

Many datasets come with a natural relational structure, that is, a graph, that contains a wealth of information about the data itself, and many data mining tasks can be accomplished from this information alone (e.g., detecting outlier elements, identifying interest groups, estimating measures of importance, and so on). Often, such tasks can be solved through suitable (sometimes, variants of standard) graph algorithms which typically assume that the graph is stored into core memory. However, this assumption is far from trivial when large graphs are dealt with; for example, the Facebook graph has more than 2 billion nodes

and hundreds of billions of edges. And while it is difficult to provide reliable estimates of the size of the web, search engines index dozens of billions of pages and consequently manage graphs with trillions of arcs.

Finding effective techniques to store and access large graphs that can be applied fruitfully to these situations is one of the central algorithmic issues in the field of modern data mining. A *compressed data structure* for a graph must provide very fast access to the graph (let us say, slower but comparable to the access time required by its uncompressed representation) *without decompressing* it; on the contrary, a *data compression scheme* for a graph is a compressed representation whose only evaluation criterion is the compression ratio (in the case of a graph, the number of bits per arc). While this definition is not formal, it clearly excludes methods in which the successors of a node are not accessible unless, for instance, a large part of the graph is decompressed. Research has concentrated on compressed data structures, as data compression schemes for graphs are relatively easy to design using standard compression techniques.

When a compressed data structure for graphs is evaluated, the following aspects should be considered:

- The *compression time*, that is, the time required to produce a compressed representation from an uncompressed one.
- Whether the data structure is *static* or *dynamic*, that is, whether it allows for changes (addition/deletion of nodes/arcs).
- Whether the data structure is designed for *directed* or *undirected* graphs (or both).
- The *compression ratio*, that is, the ratio between the size of the compressed data structure and its uncompressed counterpart, measured in bits per arc.
- The *access primitives* that the structure allows for and their corresponding performance. The four basic operations are sequential enumeration of all arcs (usually in lexicographical order), enumeration of the successors of a node x (i.e., enumerating the nodes y such

that $x \rightarrow y$), enumeration of the predecessors of a node x (i.e., enumerating the nodes y such that $y \rightarrow x$), and adjacency test between two nodes x and y (i.e., determining whether $x \rightarrow y$).

In most cases, these aspects can only be evaluated experimentally on a certain number of datasets, although in some rare circumstances, it is possible to provide worst-case lower bounds under some assumption on the network structure (e.g., assuming some probabilistic or deterministic graph model and evaluating the compression performances on that model).

WebGraph Compression

A pioneering attempt at compressing the WebGraph is the LINK database (Randall et al. 2001), where the authors exploit two basic observations: suppose that nodes are ordered lexicographically by URL (i.e., node i is the node representing the i -th URL in lexicographic order); then the following two properties are true:

- *Locality*: most arcs are between nodes that are close to each other in the order, because most links are intra-site, and URLs from the same site share a long prefix, which makes them close in lexicographical order. Locality can be exploited using *gap compression*: if node x has successors y_1, y_2, \dots, y_k , gap compression stores for x the compressed successor list $y_1, y_2 - y_1, \dots, y_k - y_{k-1}$; by locality, most values in this list will be small. Then, any variable-length encoding that uses fewer bits for small values will then provide compression: in Randall et al. (2001), the authors use variable-length nibble codes.
- *Similarity*: nodes that are close to each other in the order tend to have similar sets of neighbors. Similarity can be exploited by using *reference compression*: some successor lists are represented as a difference with respect to the successor list of a previous nearby node. In

other words, if node x has a list of successors that is “similar enough” to the successor list of node $x - r$, we just refer to that reference node, store a bitmap which selects which successors of $x - r$ must be copied, and write the remaining successors explicitly.

The authors of Randall et al. (2001), in their experiments on a real-world web crawl, state that the two techniques together provide a compression of about 5.44 bits/arc for the actual graph and 5.27 for its transpose (i.e., the graph obtained by reversing all arcs). Naturally, while gap compression does not impact much on compression or access time, reference compression can (and does) increase compression and access time, especially for random-access.

Some years later, building on the same approach, the WebGraph framework (Boldi and Vigna 2004) attained using the BV scheme a compression of less 3 bits/arc (even less than 2 for the transpose graph) with a random-access successor enumeration time of a few hundreds of nanoseconds per arc (and much less than that for sequential access). Present timings are in the range of a few dozens nanoseconds per arc.

The BV scheme introduced a more aggressive reference compression, by using inclusion-exclusion blocks instead of bitmaps, and the idea of *intervalizing* successor lists by writing separately segments of consecutive integers. The latter idea is essential for the compression of a transposed WebGraph, as in a graph with strong similarity, the same node x appears in many consecutive successor lists: when we transpose the graph, this feature becomes an interval of consecutive nodes in the successor list of x . In other words, similarity and intervalization are complementary features, exchanged by transposition, and should be always considered together.

Compression methods based on the principles described above can be applied to every graph; in particular, gap compression is always effective, as it reduces the number of bits required to write a successor. However, the effectiveness of similarity is limited to graphs in which successor lists

of nodes with close identifiers have a nontrivial intersection.

Note that any approach based on gaps and similarity can only provide successor lists and sequential enumeration. Obtaining predecessors requires storing the transpose graph, too, and answering an adjacency query requires scanning the successor list of the source node. Additional data structures may be added to make this search faster but at the expense of the compression ratio. Moreover, these approaches exploit data structures that cannot be updated easily.

k^2 -Trees

A complementary and, in some sense, more general approach is that of k^2 -trees (Brisaboa et al. 2014). In this case, rather than representing the successor list, we aim at representing compactly the *adjacency matrix* of the graph, exploiting its sparseness. A k^2 -tree subdivides recursively the adjacency matrix in k^2 squares. For each level of recursion, we look at each square: if it contains at least a one, we write a one, otherwise we write a zero. When we consider lower levels, we do not write information for subsquares of a square for which we wrote a zero. The leaf containing a specific cell can be found quickly using constant-time *rank* operations, which compute the number of ones up to a given position in a bit vector (see the reference above).

In general, in k^2 -trees the access time to a successor list is an order of magnitude slower than approaches based on gap and similarity. Nonetheless, they provide predecessors, successors, and adjacency queries in the same data structure. Moreover, they can be made dynamic with a reasonable increase in access time and storage (Brisaboa et al. 2017), which would be hardly possible with gap-based approaches.

Compression-Friendly Orderings

The techniques described above are strongly sensitive to the numbering of the nodes: this is not

a big issue when applied to WebGraphs, because the lexicographical ordering of URLs is available, but makes it difficult to apply the same techniques to networks (e.g., social networks) that do not have similarly meaningful node identifiers.

The authors of the LINK database showed that a useful intrinsic ordering is obtained by sorting the nodes in lexicographical order of their row in the adjacency matrix. Even better, one can use a Gray code (Boldi et al. 2010), that is, an ordering of the n -bit strings in which adjacent strings differ by just one bit.

A surprisingly effective approach to ordering is simply that of enumerating nodes following a *visit*: in particular, a breadth-first visit (Apostolico and Drovandi 2009) numbers nodes in such a way that an *ad hoc* compression scheme based on gap compression and reference to previous nodes provides excellent compression and very fast access times.

The problem of finding compression-friendly orderings was studied from a theoretical viewpoint in Chierichetti et al. (2013), where the authors show that the problem of determining the optimal renumbering of nodes is NP-hard but propose a heuristic (called “shingle ordering”) for the problem, based on a fingerprint of the out-neighborhoods.

A set of different approaches is based on *clustering*: *layered labelled propagation* (Boldi et al. 2011) is a very scalable reordering technique which combines the information from a number of clusterings to reorder the nodes in a very effective way. More recently, Dhulipala et al. (2016) extended the theoretical model developed by Chierichetti et al. (2013) and designed a novel ordering technique, called *recursive graph bisection*, that yields in several cases the most competitive compression ratios. The basic idea is to divide recursively the graph in two clusters of the same size so to minimize an objective function that estimates the compressibility of the overall graph when nodes in the first cluster have smaller identifiers than nodes in the second cluster. Both methods then rely on the BV scheme from WebGraph to perform compression.

Succinct Approaches

Since the successor lists are increasing sequences of numbers, one can simply use a *succinct data structure* (Navarro 2016) to store them. Technically, a succinct data structure does not *compress* data – simply, it represents it using the minimum possible number of bits. More precisely, if an object requires z bits to be represented, a succinct data structure uses $z + o(z)$ bit. For example, there is a succinct representation of binary trees using $2n + o(n)$ bits (Jacobson 1989) for a tree with n nodes.

Nonetheless, the practical effect is that of reducing sensibly the storage. Succinct approaches are particularly useful when the graph has no evident structure (as in that case, reference compression does not really help) and when reordering the nodes is not possible (as the compression obtained is agnostic with respect to the distribution of gaps and to similarity).

A simple, practical data structure of this kind is the *Elias–Fano representation of monotone sequences* (Elias 1974). This representation is technically not succinct and in fact usually named *quasi-succinct*, but it lends itself to very efficient practical implementations.

An advantage of the Elias–Fano representation is the possibility of finding in constant time both the k -th successor and the first successor greater than or equal a given bound b . In particular, the last property makes it possible to compute adjacency in constant time and neighborhood intersections in sublinear time.

Partitioned Elias–Fano (Ottaviano and Venturini 2014) is a improvement over the basic Elias–Fano representation: given a number k of parts, an algorithm splits the successor lists in k blocks in such a way that the space used is minimized. In this way, dense blocks as those generated by similarity can be compressed efficiently, with a constant additional cost for the access operations.

We remark that using a separate succinct encoding for each successor list can be more efficient than encoding succinctly the entire graph (e.g., using an Elias–Fano list to represent the

positions of the ones in the adjacency matrix or using truly succinct representations Farzan and Munro 2013). This happens because by Jensen’s inequality the average cost *per successor* is necessarily lower. However, if one takes also into account the constant costs associated to storing each successor list, there is a trade-off that has to be evaluated on a case-by-case basis.

The Elias–Fano representation has been recently made dynamic (Pibiri and Venturini 2017) when the universe is polynomial in the list size, which makes it possible to use it in the case of dynamic graph representation.

Metadata

Recent work (Ferragina et al. 2015) has extended the idea of compression so to include strings labeling the nodes, with a corresponding enlargement in the set of available queries, which include search by label prefix on successor lists and on nodes at distance two.

Software Libraries

- The WebGraph framework: <http://webgraph.di.unimi.it/>
- Apostolico–Drovandi BFS compression: <https://github.com/drovandi/GraphCompressionByBFS>

Cross-References

- ▶ [Inverted Index Compression](#)

References

- Apostolico A, Drovandi G (2009) Graph compression by BFS. *Algorithms* 2(3):1031–1044. <https://doi.org/10.3390/a2031031>
- Boldi P, Vigna S (2004) The WebGraph framework I: compression techniques. In: Proceedings of the thirteenth international world wide web conference (WWW 2004). ACM Press, Manhattan, pp 595–601

- Boldi P, Santini M, Vigna S (2010) Permuting web and social graphs. *Internet Math* 6(3):257–283
- Boldi P, Rosa M, Santini M, Vigna S (2011) Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks, pp 587–596. <https://doi.org/10.1145/1963405.1963488>
- Brisaboa N, Ladra S, Navarro G (2014) Compact representation of WebGraphs with extended functionality. *Inf Syst* 39(1):152–174. <https://doi.org/10.1016/j.is.2013.08.003>
- Brisaboa N, Cerdeira-Pena A, de Bernardo G, Navarro G (2017) Compressed representation of dynamic binary relations with applications. *Inf Syst* 69:106–123. <https://doi.org/10.1016/j.is.2017.05.003>
- Chierichetti F, Kumar R, Lattanzi S, Panconesi A, Raghavan P (2013) Models for the compressible web. *SIAM J Comput* 42(5):1777–1802. <https://doi.org/10.1137/120879828>
- Dhulipala L, Kabiljo I, Karrer B, Ottaviano G, Pupyrev S, Shalita A (2016) Compressing graphs and indexes with recursive graph bisection, 13–17 Aug 2016, pp 1535–1544. <https://doi.org/10.1145/2939672.2939862>
- Elias P (1974) Efficient storage and retrieval by content and address of static files. *J Assoc Comput Mach* 21(2):246–260
- Farzan A, Munro JI (2013) Succinct encoding of arbitrary graphs. *Theor Comput Sci* 513(Supplement C):38–52. <https://doi.org/10.1016/j.tcs.2013.09.031>
- Ferragina P, Piccinno F, Venturini R (2015) Compressed indexes for string searching in labeled graphs, pp 322–332. <https://doi.org/10.1145/2736277.2741140>
- Jacobson G (1989) Space-efficient static trees and graphs. In: 30th annual symposium on foundations of computer science (FOCS'89). IEEE Computer Society Press, Research Triangle Park, pp 549–554
- Navarro G (2016) Compact data structures: a practical approach. Cambridge University Press, Cambridge
- Ottaviano G, Venturini R (2014) Partitioned Elias–Fano indexes. In: Proceedings of the 37th international ACM SIGIR conference (SIGIR'14). ACM, New York, pp 273–282. <https://doi.org/10.1145/2600428.2609615>
- Pibiri GE, Venturini R (2017) Dynamic Elias–Fano representation. In: 28th annual symposium on combinatorial pattern matching (CPM), pp 30:1–30:14. <https://doi.org/10.4230/LIPIcs.CPM.2017.30>
- Randall K, Stata R, Wickremesinghe R, Wiener JL (2001) The LINK database: fast access to graphs of the Web. Research Report 175, Compaq Systems Research Center, Palo Alto

Well-Being

► Big Data for Health

Wildfire: HTAP for Big Data

Ronald Barber¹, Vijayshankar Raman¹, Richard Sidle¹, Yuanyuan Tian¹, and Pinar Tözün²

¹IBM Research – Almaden, San Jose, CA, USA

²IT University of Copenhagen, Copenhagen, Denmark

Definitions

The popularity of large-scale real-time analytics applications keeps rising. These applications require distributed data management systems that can handle fast concurrent transactions (OLTP) and analytics on the recent data. Some of them even need running analytical queries (OLAP) as part of transactions. For the kind of data processing that requires both analytics and transactions simultaneously in the same system, Gartner recently coined the term *Hybrid Transactional/Analytical Processing (HTAP)*.

Overview

Traditional database management systems (DBMSs) have been the strongest for transactions based on the ACID properties and complex analytics over structured data where data is loaded into a closed format. In the past decade, several specialized DBMSs have emerged exploiting modern hardware (e.g., multicore parallelism, large main memories) and following leaner system designs: in-memory optimized row-stores for OLTP (Diaconu et al. 2013; Stonebraker and Weisberg 2013) and column-stores for OLAP (Boncz et al. 2005; Lamb et al. 2012; Raman et al. 2013). Even though these new-gen systems strengthen the capabilities of traditional DBMSs, they fall short in handling applications that require both transactions and different types of complex analytics as well as processing of different big data sources at a large scale.

These weaknesses of DBMSs largely motivated the development of big data ecosystems

such as Hadoop <http://hadoop.apache.org/> and Spark <http://spark.apache.org/>, which were designed almost exclusively for (1) scaling out over thousands of commodity servers and (2) performing complex and long-running analytics cost-effectively on extremely large and diverse data sets. These systems also promote a much more open and flexible environment, both of functions (e.g., external machine learning and graph processing libraries) and standard open data formats (e.g., Parquet, JSON), allowing any function to access any data without the obligation of going through a centralized gatekeeper.

However, these big data ecosystems have their own shortcomings. Since platforms like Spark are heavily optimized for big data analytics, data ingest and point lookup requests are relegated to external systems (e.g., with connectors to key-value stores, DBMSs). Having two different systems for transactions and analytics, however, delays the time it takes to query recently ingested data. In addition, most of the systems that can handle the high ingest rates required to capture data from new Internet of Things (IoT) applications (e.g., key-value stores such as Cassandra <http://cassandra.apache.org>, Aerospike <http://www.aerospike.com/>), lack support for complex transactions (multi-statement/row), and ACID.

To serve the needs of the emerging applications that require both complex transactions and analytics at large scale in a unified platform over the latest, almost latest, and historical data, there is a need to build systems that can handle hybrid transactional and analytical processing (HTAP) efficiently. In addition to the challenge of supporting HTAP, these systems must handle the evolution of the data from the time it gets ingested to the time it becomes historical as the data lies over several storage mediums (memory, SSD, HDD, shared file systems, local or remote storage) and possibly in different data formats (row, column, hybrid).

This article presents Wildfire (Barber et al. 2017), an HTAP system for big data. Wildfire exploits Spark by (1) utilizing a non-proprietary storage format (Parquet), open to any reader, for all data, (2) using and extending Spark APIs and the Catalyst optimizer for SQL queries,

and (3) automatically replicating data for high availability, scale-out performance, and elasticity. To enable HTAP, Wildfire augments these features via (1) making the latest committed data immediately available to analytics queries; (2) having support for indexes for fast point queries; (3) exploiting recent advances for accelerating analytics queries by orders of magnitude, including compression on the fly, cache-aware processing, automatic creation and exploitation of synopses, (a form of) column-wise storage, and multi-threaded parallelism; and (4) enterprise-quality SQL, including more robust optimization and time travel that permits querying historical data AS OF a particular time.

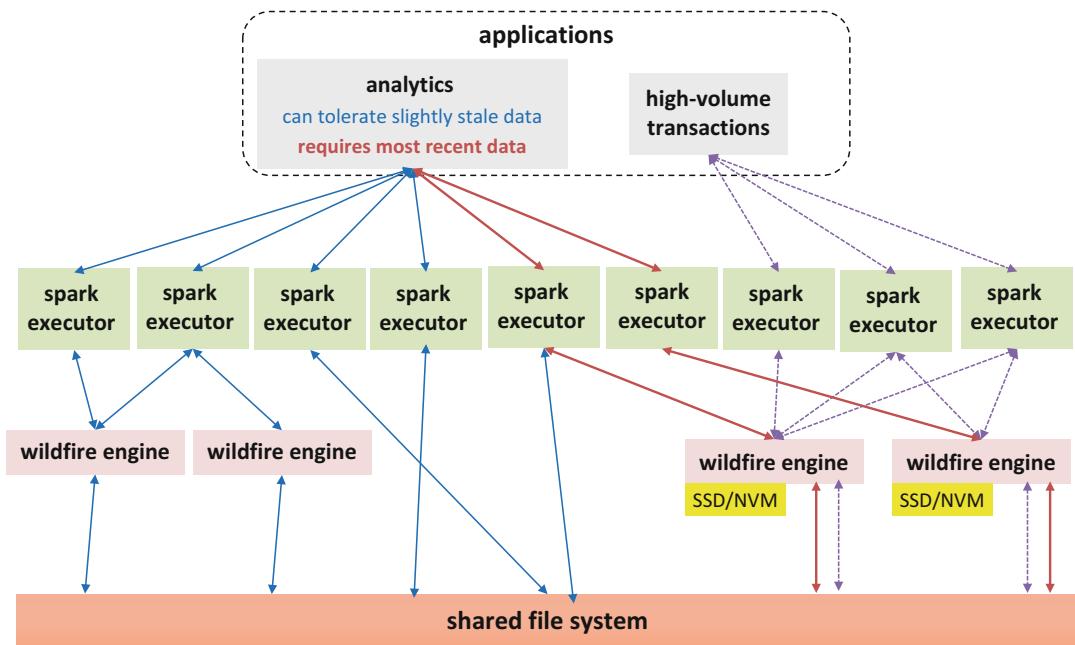
Wildfire Architecture

Figure 1 shows the Wildfire architecture, which combines the Spark ecosystem with the Wildfire engine. Spark serves as the entry point for all the requests (transactions and analytics) from the applications that Wildfire targets and provides a scalable and integrated ecosystem for various types of analytics on big data. The Wildfire engine, on the other hand, processes transactions, accelerates the analytical requests, and enables querying the newly ingested data.

Processing of Requests

All requests to Wildfire go through Spark APIs. Each request spawns Spark executors across a cluster of machines whose nodes depend upon the type of that request. The majority of the nodes in the cluster execute only analytical requests and require only commodity server hardware (the solid arrows in Fig. 1 show the request and data flow in these nodes). Other beefier nodes, with faster local persistent storage (SSDs or, soon, NVRAM) and more cores for increased parallelism, handle concurrently both transactions and analytical queries on the recent data from those transactions (the dashed arrows in Fig. 1 show the request and data flow in these nodes).

The Wildfire engine is a columnar data processing engine. Each engine instance is a daemon serving on a Wildfire node. Each daemon is



Wildfire: HTAP for Big Data, Fig. 1 Data lifecycle in Wildfire

connected to a Spark executor, which delegates various transactional and analytical requests to the daemons. Based on the nodes they serve on, these daemons either handle only read-only requests over a snapshot of the data or serve transactions and analytical requests over the latest version of the data.

Transactions always go through the Wildfire engine daemons that serve on the beefier server nodes responsible for transaction processing. In addition to handling ingest, update, and lookup operations, these daemons publish the ingested data to a shared file system in an open data format (Parquet <https://parquet.apache.org/>) and can also read any data that is in the shared file system for analytical queries.

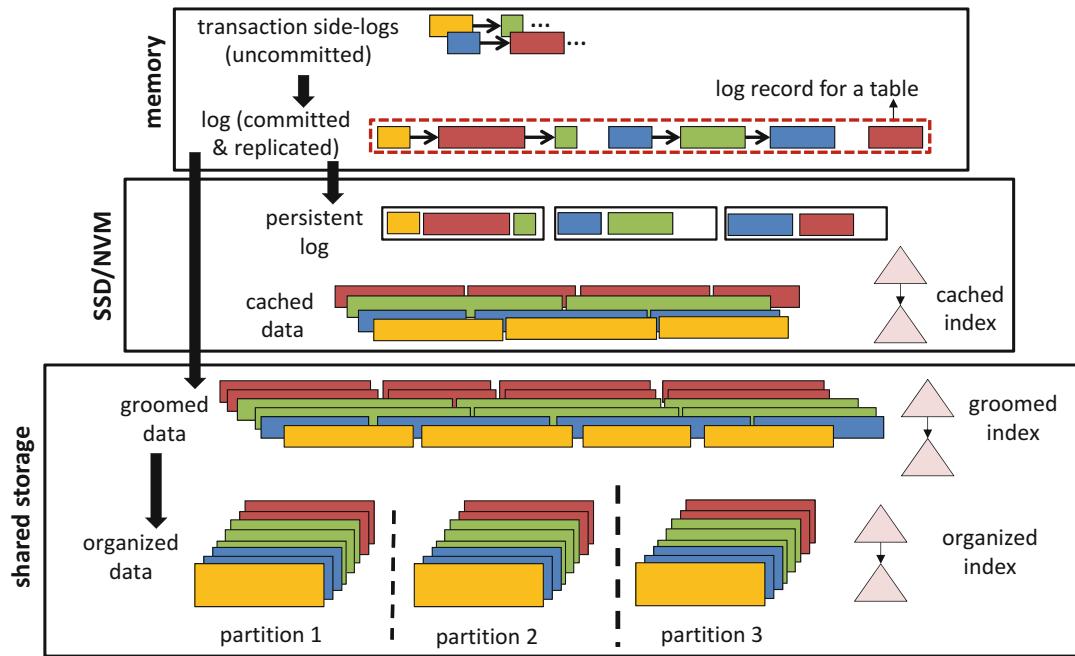
To speed ingest through parallelism, non-static tables in the system are sharded across nodes handling transactions based upon a subset of a primary (single-column or composite) key. A table shard is also assigned to (a configurable number of) multiple nodes for higher availability. A distributed coordination system manages the meta-information related to sharding and replica-

tion, and a catalog maintains the schema information for each table.

The analytical requests in Wildfire can either be handled solely by Spark or the Wildfire engine or both. Wildfire openly allows any external reader to read data ingested by the Wildfire engine using Spark APIs without involving the Wildfire engine component but that reader will be unable to see the latest transactional data. Wildfire can also push down the whole or part of analytical requests into the Wildfire engine. This way one can accelerate the processing of the analytical requests using the Wildfire columnar engine and access the latest data for analytics.

Data Life Cycle

Wildfire adds the following three hidden columns to every table to keep track of the life of each record, which allows supporting snapshot isolation and time travel queries. beginTS indicates when a record is first made available in shared storage; endTS is the time when a record is updated; prevRID is the id of the new record who replaces this record.



Wildfire: HTAP for Big Data, Fig. 2 Data lifecycle in Wildfire

Data in Wildfire evolves over time from live to groomed to organized zone and is kept in Parquet format across all zones. Figure 2 illustrates the data life cycle in a shard replica in Wildfire.

Live zone keeps the latest data and is composed of transaction logs. Each transaction in the Wildfire engine keeps its uncommitted changes in a transaction-local side-log at a shard replica. At commit time, the transaction appends its side-log to the committed-log of that shard, which is kept both in memory and persisted on the local disk (SSD or NVRAM). In addition, this log is copied to each of the other replica nodes for that shard.

Groomed zone keeps a recent (~1 s) snapshot of the data. One of the replicas of a shard hosts a groomer daemon, which periodically invokes a grooming operation periodically (every second) to move the data from the live zone to the groomed zone. This operation merges transaction logs from shard replicas based on the commit time order, resolves conflicts by setting the monotonic increasing beginTS for each record, and creates Parquet files larger than the log blocks for the corresponding table shard. These files are then flushed to

the shared storage (HDFS <http://hortonworks.com/apache/hdfs/>, GlusterFS (<https://www.gluster.org/>), S3 <https://aws.amazon.com/s3/>, Swift <https://www.swiftstack.com/product/openstack-swift>). Finally, the groomer notifies a distributed coordinator (Zookeeper <https://zookeeper.apache.org/>) about the groom high watermark and then prunes the log records that are successfully groomed. This allows us to safely switch the shard replicas that are responsible from the grooming operation (i.e., upon a failure of the node that hosts the groomer of a shard replica).

Organized zone keeps an almost recent (~10 mins) and more analytics-friendly snapshot of the data. The groomed zone (1) has still relatively small Parquet files, (2) organizes data based on the sharding key, and (3) has only the beginTS field set among the hidden columns of the records. In order to evolve the data into a more analytics-friendly state, a separate operation called post-groom takes places periodically (every 10 mins). Post-groom takes the files in the groomed zone and merges them into bigger Parquet files. It further partitions the data in a

shard if the user/application specifies a partition key as part of a tables' definition to accelerate common analytics tasks. For example, an IoT application handling large volumes of sensor readings could use the sensorID as the sharding key but the date column as the partition key to speed up time-based analytics queries. Finally, it also fills up the endTS and prevRID fields of the records to enable time travel queries and faster snapshot isolation. To optimize the data organization further for the analytics, the latest versions of the records are also separated from the records where the endTS is set.

Indexes in Wildfire are an LSM-like (O’Neil et al. 1996) structure with a list of index runs, where each groom operation produces a new index run. Runs are organized into levels and are merged over time to limit the total number of runs and improve lookup performance. Index instances are associated with each table shard and are built over the groomed and organized zones. There are no indexes over the live zone to not to impact ingest rate. The data in live zone is very small since grooming runs very frequently, and one can just scan this zone to find the items they need.

Finally, Wildfire assumes the recently ingested data is accessed more often by the applications. Therefore, the recently groomed data files and their corresponding indexes are also cached in the local storage of the nodes based on the space utilization.

Transactions

In addition to being a columnar query accelerator for the Spark ecosystem, the Wildfire engine also supports transactions with inserts, updates, and deletes. Wildfire targets both high availability and ACID, which is infeasible. Therefore, Wildfire adopts last-writer-wins (LWW) semantics for concurrent updates to the same key and snapshot isolation of quorum-readable content for queries, without having to read the data from a quorum of replicas to satisfy a query. The remainder of this section describes the design choices and methods to reach this goal.

Writes, Inserts, Updates, and Deletes

A table in Wildfire is sharded across nodes using a subset of the primary key columns of the table. The writes of a transaction are sent to any node that contains a shard replica. As section “[Data Life Cycle](#)” describes, Wildfire first writes the transactional changes to logs persisted in local storage, since it is impractical to flush them to the shared file system directly. For high availability, these shard logs are replicated to multiple nodes (a minimum of three) at commit time. Then, a background grooming process propagates these changes in a batched fashion to the shared file system, which is optimized for append-only access and large blocks.

Transactions with write operations have to perform a status-check at the end in Wildfire to ensure that the side-log of the transaction is replicated at least to a quorum of nodes. Similarly, the read-only queries return only quorum-replicated data. The status-check may timeout if a node is poorly connected. In such cases, Wildfire returns a warning message to the client about the uncertainty of the status of that transaction. Due to this delayed-commit semantics, one cannot check integrity constraints at commit. Updates to the same key based on prior values may suffer from the lost update problem. Wildfire resolves this by adopting LWW as mentioned above.

Wildfire also offers a SyncWrite option for the applications with idempotent writes, where it reissues any timed out transaction on other nodes till they succeed. If a non-idempotent write times out or the client connection breaks, there is no easy way to figure out whether that write has succeeded.

The conflict resolution in Wildfire is handled with the collaboration of the grooming and the post-grooming operations. The post-grooming operation sets the endTS of a record with the beginTS of the next version of that record, as well as the prevRID field of a record to the record id of the previous version of that record. Post-grooming heavily relies on the indexes built on the primary key of a table while determining the correct values of for these fields efficiently. The grooming operation sets the beginTS fields

of records and builds the corresponding index blocks over the data it grooms adding them as part of the top level of the Wildfire index structure, which resembles an LSM-based index. After the post-grooming operation, the corresponding portions of the index are replaced asynchronously to ensure that records point to the right Parquet blocks.

Reads

Snapshot isolation needs a system-generated predicate: $\text{beginTS} \leq \text{snapshotTS} < \text{endTS}$. The snapshotTS is usually the transaction start time but can be changed to allow time travel. Depending on the currency of data needed by queries in Wildfire, data in the shared file system may be sufficient. However, certain classes of queries read the log along with the data in the shared file system, whereas grooming itself (treated as a specialized query in Wildfire) reads only the quorum-visible log to perform its processing.

Queries that only read the organized zone do not have to perform additional conflict resolution to determine the correct version of the records, since post-groom already handles that. Queries that have to read the groomed zone might find unresolved duplicates, on the other hand, and have to perform an index lookup over the primary index to determine the correct version under LWW. Finally, queries that require reading the live zone can do so without accessing all the replicas, since each instance of Wildfire tracks the log replication points for all replicas it is responsible for and computes a current high watermark of the data that is quorum-visible.

To not to disrupt the older queries that are still reading the log blocks that have been groomed or the groomed blocks that have been post-groomed, Wildfire performs lazy garbage collection of such blocks. In other words, data blocks are garbage collected only when there is no possibility of a query reading them. When a new query has to access multiple data zones, though, Wildfire checks the groom and post-groom high watermarks to ensure that it does not read duplicate data streams.

Analytics

This section describes the major extensions in Spark for Wildfire: (1) the new OLTP interface OLTPContext, (2) extensions to the Spark Catalyst optimizer and the existing OLAP SQLContext to enable the push-down of queries into the Wildfire engine, and (3) using these two contexts to enable HTAP in Wildfire.

New Interface for OLTP

To enable HTAP, Wildfire has to support functionality required for OLTP operations such as point queries, inserts, or upserts. However, the Spark ecosystem, which is the entry point for applications running on Wildfire, does not currently support such functionality. Therefore, Wildfire implements a new interface in Spark called OLTPContext. OLTPContext is separated from Sparks SQLContext but plays well with the existing APIs of Spark such as Spark SQL and Spark Streaming.

OLTPContext has to figure out the right set of nodes to send the specific OLTP requests to. Therefore, it contacts the distributed coordination service to retrieve and cache the information regarding where each table shard replica is hosted as well as the reference to the catalog service. Based on this information, if a sharding key is associated with a transaction (or a batch of transactions), OLTPContext is able to route the request(s) to the right node. If no sharding key is given as part of transaction, the request is broadcasted to all shards.

OLTPContext also handles node failures. If a transaction fails due to the failure of a node that is responsible from a shard replica for the shard corresponding to that transaction, then OLTPContext will retry that transaction on one of the other nodes who host another replica of that shard.

W

Extensions to SparkSQL for OLAP

Wildfire aims a seamless integration of Spark SQL and the Wildfire engine for analytics queries. Users should be able to query the tables in Wildfire using Spark SQL and join a table in Wildfire with an external table accessible by Spark.

Wildfire achieves this integration by extending both Spark SQL Data Source API and the Catalyst query optimizer. It is easy to plug a new data source using the Data Source API through Spark SQL. Wildfire just exploits this and extends Spark's SQLContext. The Catalyst optimizer, on the other hand, is able to push down projection and filtering operations to the data sources. The Wildfire engine, however, provides more advanced query capabilities. Therefore, Wildfire extends the Catalyst optimizer to push down more complex analytics operations such as partial aggregations, joins, and even scalar user-defined functions and aggregates (by efficiently utilizing the JNI interface). Wildfire's complex push-down analysis is also able to generate compensation plans for the remaining portions of the analytics queries that cannot be pushed down into Wildfire's columnar engine.

Finally, Wildfire's extensions to the Data Sources API and the Catalyst optimizer are general and not just for Wildfire. Therefore, this general method of push-down enables Spark to be a federation engine for big data systems.

Using OLTPContext and SQLContext for HTAP

To utilize Wildfire's HTAP capabilities, applications should instantiate both the new OLTPContext and the SQLContext with the Wildfire extensions in the Spark driver. The transactions and point queries go through the OLTPContext. SQLContext handles the analytical queries, and assigns a snapshot to them based on the queries maximum tolerable staleness of the data. If this staleness is less than the grooming interval, the query is either delayed till the next grooming cycle for the corresponding shard(s) or sent to the Wildfire nodes with the latest data (logs) for the corresponding shard(s) in their local storage. The latter case may have negative impact on the transaction throughput on nodes assigned to OLTP tasks due to resource consumption of the analytical tasks. This is, however, similar to the resource management challenges of the traditional database systems that need to handle both transactions and analytics. Wildfire can process the queries that can tolerate a staleness

longer than the grooming or post-grooming interval more inexpensively, since such queries can be sent to any Wildfire node and access the data from the shared file system.

Conclusions

This article presented the Wildfire HTAP system, which is a distributed big data platform designed to handle high-volume transactions and complex analytics over the latest and almost latest data concurrently. Wildfire leverages Spark for large-scale complex analytics, adopts an open data format to be compatible with the big data ecosystem, and implements a columnar processing engine that supports transactions and accelerates the analytical queries that are issued through the SparkSQL API.

Cross-References

- [Hybrid OLTP and OLAP](#)
- [SnappyData](#)

References

- Barber R, Garcia-Arellano C, Grosman R, Mueller R, Raman V, Sidle R, Spilchen M, Storm AJ, Tian Y, Tözün P et al (2017) Evolving databases for new-gen big data applications. In: CIDR
- Boncz P, Zukowski M, Nes N (2005) MonetDB/X100: hyper-pipelining query execution. In: CIDR
- Diaconu C, Freedman C, Ismert E, Larson P-Å, Mittal P, Stonecipher R, Verma N, Zwilling M (2013) Hekaton: SQL server's memory-optimized OLTP engine. In: SIGMOD, pp 1243–1254
- Lamb A, Fuller M, Varadarajan R, Tran N, Vandiver B, Doshi L, Bear C (2012) The vertica analytic database: C-store 7 years later. PVLDB 5(12):1790–1801
- O'Neil P, Cheng E, Gawlick D, O'Neil E (1996) The log-structured merge-tree (LSM-tree). Acta Inf 33(4): 351–385
- Raman V, Attaluri G, Barber R, Chainani N, Kalmuk D, KulandaiSamy V, Leenstra J, Lightstone S, Liu S, Lohman GM, Malkemus T, Mueller R, Pandis I, Schiefer B, Sharpe D, Sidle R, Storm A, Zhang L (2013) DB2 with BLU acceleration: so much more than just a column store. PVLDB 6:1080–1091
- Stonebraker M, Weisberg A (2013) The VoltDB main memory DBMS. IEEE Data Eng Bull 36(2):21–27

Workflow Audit Trail

- ▶ Business Process Event Logs and Visualization

Workflow Log

- ▶ Business Process Event Logs and Visualization

Workflow Mining

- ▶ Business Process Analytics

Workflow Systems for Big Data Analysis

Loris Belcastro and Fabrizio Marozzo
DIMES, University of Calabria, Rende, Italy

Definitions

A *workflow* is a well-defined, and possibly repeatable, pattern or systematic organization of activities designed to achieve a certain transformation of data (Talia et al. 2015).

A *Workflow Management System* (WMS) is a software environment providing tools to define, compose, map, and execute workflows.

Overview

The wide availability of high-performance computing systems has allowed scientists and engineers to implement more and more complex applications for accessing and analyzing huge amounts of data (Big Data) on distributed and high-performance computing platforms. Given the variety of Big Data applications and types

of users (from end users to skilled programmers), there is a need for scalable programming models with different levels of abstractions and design formalisms. The programming models should adapt to user needs by allowing (i) ease in defining data analysis applications, (ii) effectiveness in the analysis of large datasets, (iii) and efficiency of executing applications on large-scale architectures composed by a massive number of processors. One of the programming models that meets these requirements is the workflow, which through its convenient design approach has emerged as an effective paradigm to address the complexity of scientific and business Big Data analysis applications.

According to the definition of the Workflow Management Coalition (WFMC 1999), a workflow is “*the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*” The term “process” here indicates a set of tasks, or activities, linked together with the goal of creating a product, calculating a result, providing a service and so on. Hence, each task represents a piece of work that forms one logical step of the overall process (Georgakopoulos et al. 1995). The same definition can be used for scientific workflows composed of several tasks (or activities) that are connected together to express data and/or control dependencies (Liu et al. 2004). Thus, a workflow can be also defined as a “*well defined, and possibly repeatable, pattern or systematic organization of activities designed to achieve a certain transformation of data*” (Talia et al. 2015).

From a practical point of view, a workflow consists of a series of tasks, activities, or events that must be performed to accomplish a goal and/or obtain a result. For example, a data analysis workflow can be designed as a sequence of preprocessing, analysis, post-processing, and evaluation tasks. It can be implemented as a computer program and can be expressed in a programming language for describing the workflow tasks and mechanisms to orchestrate them. Important benefits of the workflow formalism are the following: (i) it provides a declarative way for

specifying the high-level logic of an application, hiding the low-level details that are not fundamental for application design; (ii) it is able to integrate existing software modules, datasets, and services in complex compositions that implement scientific discovery processes; (iii) and once defined, workflows can be stored and retrieved for modifications and/or re-execution, by allowing users to define typical patterns and reuse them in different scenarios (Bowers et al. 2006).

A Big Data analysis workflow can be designed through a script- or a visual-based formalism. The *script-based formalism* offers a flexible programming approach for skilled users who prefer to program their workflows using a more technical approach. Moreover, this formalism allows users to program complex applications more rapidly, in a more concise way, and with higher flexibility (Marozzo et al. 2015). In particular, script-based applications can be designed in different ways: (i) with a programming language that allows to define tasks and dependencies among them, (ii) with annotations that allow the compiler to identify which instructions will be executed in parallel, and (iii) using a library in the application code to define tasks and dependencies among them. As an alternative, the *visual-based formalism* is a very effective design approach for high-level users, e.g., domain expert analysts having a limited knowledge of programming languages. A visual-based application can be created by using a visual programming language that lets users develop applications by programming the workflow components graphically. A visual representation of workflows intrinsically captures parallelism at the task level, without the need to make parallelism explicit through control structures (Maheshwari et al. 2013).

Key Research Findings

To cope with the need of high-level tools for the design and execution of Big Data analysis workflows, in the past years, many efforts have been made for the development of distributed Workflow Management Systems (WMSs), which are devoted to support the definition, creation, and

execution of workflows. A WMS is a software environment providing tools to define, compose, map, and execute workflows. A key function of a WMS during the workflow execution is co-ordinating the operations of individual activities that constitute the workflow. There are several WMSs on the market, most of them targeted to a specific application domain. Existing WMSs can be grouped roughly into two main classes:

- *Script-based systems*, which permit to define workflow tasks and their dependencies through instructions of traditional programming languages (e.g., Perl, Ruby, or Java) or custom-defined languages. Such languages provide specific instructions to define and execute workflow tasks, such as sequences, loops, while-do, or parallel constructs. These types of instructions declare tasks and their parameters using textual specifications. Typically data and task dependencies can be defined through specific instructions or code annotations. Examples of script-based workflow systems are Swift (Wilde et al. 2011), COMPSs (Lordan et al. 2014) and DMCF (Marozzo et al. 2015).
- *Visual-based systems*, which allows to define workflows as a graph, where the nodes are resources and the edges represent dependencies among resources. Compared with script-based systems, visual-based systems are easier to use and more intuitive for domain-expert analysts having a limited understanding of programming. Visual-based workflow systems often incorporate graphical user interfaces that allow users to model workflows by dragging and dropping graph elements (e.g., nodes and edges). Examples of visual-based systems are Pegasus (Deelman et al. 2015), CloudFlows (Kranjc et al. 2012), and Kepler (Ludäscher et al. 2006).

Another classification can be done according to the way a workflow is represented. Although a standard workflow language like Business Process Execution Language (BPEL) (Juric et al. 2006) has been defined, scientific workflow

systems often have developed their own workflow representations. Other than BPEL, other formalisms are used to represent and store workflows, such as JSON (Marozzo et al. 2015), Petri nets (Guan et al. 2006), and XML-based languages (Atay et al. 2007). This situation makes difficult sharing workflow codes and limits interoperability among workflow-based applications developed by using different workflow management systems. Nevertheless, there are some historical reasons for that, as many scientific workflow systems and their workflow representations were developed before BPEL existed (Margolis 2007).

In the following, we present some representative example of workflow systems that can be used to implement applications for Big Data analysis. Some of them have been implemented on parallel computing systems, others on Grids; recently some have been made available on Clouds.

Swift (Wilde et al. 2011) is a system for creating and running workflows across several distributed systems, like clusters, Clouds, Grids, and supercomputers. Swift is based on a C-like syntax and uses an implicit data-driven task parallelism (Wozniak et al. 2014). In fact, it looks like a sequential language, but all variables are *futures*; thus the execution is based on data availability (i.e., when the input data is ready, functions are executed in parallel).

COMPSS (Lordan et al. 2014) is another workflow system that aims at easing the development and execution of workflows in distributed environments, including Grids and Clouds. With COMPSS, users create a Java sequential application and select which methods will be executed remotely by providing annotated interfaces. The runtime intercepts any call to a selected method creating a representative task and finding the data dependencies with all the previous ones that must be considered along the application run.

Data Mining Cloud Framework (DMCF) (Marozzo et al. 2016) is a software system for designing and executing data analysis workflows on Clouds. A workflow in DMCF can be developed using a visual- or a script-based language. The visual language, called VL4Cloud (Marozzo et al. 2016), is based on

a design approach for end users having a limited knowledge of programming paradigms. The script-based language, called JS4Cloud (Marozzo et al. 2015), provides a flexible programming paradigm for skilled users who prefer to code their workflows through scripts. Both languages implement a data-driven task parallelism that spawns ready-to-run tasks to Cloud resources.

Pegasus (Deelman et al. 2015) is a workflow management system developed at the University of Southern California for supporting the implementation of scientific applications also in the area of data analysis. Pegasus includes a set of software modules to execute workflow-based applications in a number of different environments, including desktops, Clouds, Grids, and clusters. It has been used in several scientific areas including bioinformatics, astronomy, earthquake science, gravitational wave physics, and ocean science.

CloudFlows (Kranjc et al. 2012) is a Cloud-based platform for the composition, execution, and sharing of interactive data mining workflows. It provides a user interface that allows programming visual workflows in any Web browser. In addition, its service-oriented architecture allows using third-party services (e.g., Web services wrapping open-source or custom data mining algorithms). The server side consists of methods for the client side workflow editor to compose and execute workflows and a relational database of workflows and data.

Microsoft Azure Machine Learning (Azure ML) (<https://azure.microsoft.com/en-us/services/machine-learning-studio/>) is a SaaS that provides a Web-based machine learning IDE (i.e., Integrated Development Environment) for creation and automation of machine learning workflows. Through its user-friendly interface, data scientists and developers can perform several common data analyses/mining tasks on their data and automate their workflows. Using its drag-and-drop interface, users can import their data in the environment or use special readers to retrieve data from several sources, such as Web URL (HTTP), OData Web service, Azure Blob Storage, Azure SQL Database, and Azure Table.

Taverna (Wolstencroft et al. 2013) is a workflow management system developed at

the University of Manchester. Its primary goal is supporting the life sciences community (biology, chemistry, and medicine) to design and execute scientific workflows and support in silico experimentation, where research is performed through computer simulations with models closely reflecting the real world. Even though most Taverna applications lie in the bioinformatics domain, it can be applied to a wide range of fields since it can invoke any REST- or SOAP-based Web services.

Kepler (Ludäscher et al. 2006) is a graphical workflow management system that has been used in several projects to manage, process, and analyze scientific data. Kepler provides a graphical user interface (GUI) for designing scientific workflows, which are a structured set of tasks linked together that implement a computational solution to a scientific problem.

Examples of Application

Workflows are widely used by scientists to acquire and analyze huge amount of data for complex analysis, such as in physics (Brown et al. 2007), medicine (Lu et al. 2006), and sociology (Marin and Wellman 2011).

The Pan-STARRS astronomical survey (Deelman et al. 2009) used Microsoft Trident Scientific Workflow Workbench for loading and validating telescope detections running at about 30 TB per year. Similarly, the USC Epigenome Center is currently using Pegasus for generating high-throughput DNA sequence data (up to eight billion nucleotides per week) to map the epigenetic state of human cells on a genome-wide scale (Juve et al. 2009). The Laser Interferometer Gravitational Wave Observatory (LIGO) uses workflows to design and implement gravitational wave data analysis, such as the collision of two black holes or the explosion of supernovae. The experiment records approximately 1 TB of data per day, which is analyzed by scientists in all parts of the world (Brown et al. 2007). In this scenario, workflow formalism demonstrates its effectiveness in programming Big Data scientific applications.

The workflow formalism has been also used for implementing and executing complex data mining applications on large datasets. Some examples are *Parallel clustering* (Marozzo et al. 2011), where multiple instances of a clustering algorithm are executed concurrently on a large census dataset to find the most accurate data grouping; *Association rule analysis* (Agapito et al. 2013), which is a workflow for association rule analysis between genome variations and clinical conditions of a group of patients; and *Trajectory mining* (Altomare et al. 2017) for discovering patterns and rules from trajectory data of vehicles in a wide urban scenario. In some cases, the workflow formalism has been integrated with other programming models, such as MapReduce, to exploit the inherent parallelism of the application in presence of Big Data. As an example, in Belcastro et al. (2015), a workflow management system has been integrated with MapReduce for implementing a scalable predictor of flight delays due to weather conditions (Belcastro et al. 2016).

Future Directions for Research

Workflow systems for Big Data analysis require high-level and easy-to-use design tools for programming complex applications dealing with huge amount of data. There are several open issues that will require research and development in the near future, such as:

- *Programming models.* Several scalable programming models have been proposed, such as MapReduce (Dean and Ghemawat 2008), Message Passing (Gropp et al. 1999), Bulk Synchronous Parallel (Valiant 1990), Dryad (Isard et al. 2007), or Pregel (Malewicz et al. 2010). Some development works must be done for extending workflow systems to support different programming models, which could improve their capabilities in terms of efficiency, scalability, and interoperability with other systems.
- *Data storage.* The increasing amount of data generated every day needs even more scal-

- able data storage systems. Workflow systems should improve their capabilities to access data stored on high-performance storage systems (e.g., NoSQL systems, Object based storage on Clouds) by using different protocols.
- *Data availability.* Workflow systems have to deal with the problem of granting service and data availability, which is an opened challenge that can negatively affect performances. Several solutions have been proposed for improving exploitation, such as using a cooperative multi-Cloud model to support Big Data accessibility in emergency cases (Lee et al. 2012), but more studies are still needed to handle the continuous increasing demand for more real-time and broad network access.
 - *Local mining and distributed model combination.* As workflow-based applications often involve several local and distributed data sources, collecting data to a centralized server for analysis is not practical or, in some cases, possible. Scalable workflow systems for data analysis have to enable local mining of data sources and model exchange and fusion mechanisms to compose the results produced in the distributed nodes. According to this approach, the global analysis can be performed by distributing the local mining and supporting the global combination of every local knowledge to generate the complete model.
 - *Data and tool interoperability and openness.* Interoperability is a main open issue in large-scale distributed applications that use resources such as data and computing nodes. Workflow systems should be extended to support interoperability and ease cooperation among teams using different data formats and tools.
 - *Integration of Big Data analysis frameworks.* The service-oriented paradigm allows running large-scale distributed workflows on heterogeneous platforms along with software components developed using different programming languages or tools. This feature should improve the integration between workflows and other scalable Big Data analysis software systems, such as frameworks for fine-grain in-memory data access and analysis. In such way,

it will be possible to extend workflows toward exascale computing, since exascale processors and storage devices must be exploited with fine-grain runtime models.

References

- Agapito G, Cannataro M, Guzzi PH, Marozzo F, Talia D, Trunfio P (2013) Cloud4snp: distributed analysis of SNP microarray data on the cloud. In: Proceedings of the ACM conference on bioinformatics, computational biology and biomedical informatics 2013 (ACM BCB 2013). ACM, Washington, DC, p 468. ISBN:978-1-4503-2434-2
- Altomare A, Cesario E, Comito C, Marozzo F, Talia D (2017) Trajectory pattern mining for urban computing in the cloud. *Trans Parallel Distrib Syst* 28(2):586–599. ISSN:1045-9219
- Atay M, Chebotko A, Liu D, Lu S, Fotouhi F (2007) Efficient schema-based XML-to-relational data mapping. *Inf Syst* 32(3):458–476
- Belcastro L, Marozzo F, Talia D, Trunfio P (2015) Programming visual and script-based big data analytics workflows on clouds. In: Grandinetti L, Joubert G, Kunze M, Pasquetti V (eds) Post-proceedings of the high performance computing workshop 2014. Advances in parallel computing, vol 26. IOS Press, Cetraro, pp 18–31. ISBN:978-1-61499-582-1
- Belcastro L, Marozzo F, Talia D, Trunfio P (2016) Using scalable data mining for predicting flight delays. *ACM Trans Intell Syst Technol* 8(1):1–20
- Bowers S, Ludascher B, Ngu AHH, Critchlow T (2006) Enabling scientific workflow reuse through structured composition of dataflow and control-flow. In: 22nd international conference on data engineering workshops (ICDEW’06), pp 70–70. <https://doi.org/10.1109/ICDEW.2006.55>
- Brown DA, Brady PR, Dietz A, Cao J, Johnson B, McNabb J (2007) A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis. *Workflows for e-Science*, pp 39–59
- Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Deelman E, Gannon D, Shields M, Taylor I (2009) Workflows and e-science: an overview of workflow system features and capabilities. *Futur Gener Comput Syst* 25(5):528–540
- Deelman E, Vahi K, Juve G, Rynge M, Callaghan S, Maechling PJ, Mayani R, Chen W, da Silva RF, Livny M et al (2015) Pegasus, a workflow management system for science automation. *Futur Gener Comput Syst* 46:17–35
- Georgakopoulos D, Hornick M, Sheth A (1995) An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib Parallel Databases* 3(2):119–153

- Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, vol 1. MIT press, Cambridge
- Guan Z, Hernandez F, Bangalore P, Gray J, Skjellum A, Velusamy V, Liu Y (2006) Grid-flow: a grid-enabled scientific workflow system with a petri-net-based interface. *Concurr Comput Pract Exp* 18(10):1115–1140
- Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: ACM SIGOPS operating systems review, vol 41. ACM, pp 59–72
- Juric MB, Mathew B, Sarang PG (2006) Business process execution language for web services: an architect and developer's guide to orchestrating web services using BPEL4WS. Packt Publishing Ltd, Birmingham
- Juve G, Deelman E, Vahi K, Mehta G, Berriman B, Berman BP, Maechling P (2009) Scientific workflow applications on Amazon EC2. In: 2009 5th IEEE international conference on E-science workshops. IEEE, pp 59–66
- Kranjc J, Podpečan V, Lavrač N (2012) Crowdflows: a cloud based scientific workflow platform. In: Machine learning and knowledge discovery in databases. Springer, pp 816–819
- Lee S, Park H, Shin Y (2012) Cloud computing availability: multi-clouds for big data service. In: Convergence and hybrid information technology. Springer, Heidelberg, pp 799–806
- Liu L, Pu C, Ruiz DD (2004) A systematic approach to flexible specification, composition, and restructuring of workflow activities. *J Database Manag* 15(1):1
- Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia R (2014) Servicess: an interoperable programming framework for the cloud. *J Grid Comput* 12(1):67–91
- Lu Q, Hao P, Curcin V, He W, Li YY, Luo QM, Guo YK, Li YX (2006) KDE bioscience: platform for bioinformatics analysis workflows. *J Biomed Inf* 39(4):440–450
- Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the kepler system. *Concurr Comput Pract Exp* 18(10):1039–1065
- Maheshwari K, Rodriguez A, Kelly D, Madduri R, Wozniak J, Wilde M, Foster I (2013) Enabling multi-task computation on galaxy-based gateways using swift. In: 2013 IEEE international conference on cluster computing (CLUSTER). IEEE, pp 1–3
- Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 135–146
- Marin A, Wellman B (2011) Social network analysis: an introduction. The SAGE handbook of social network analysis, p 11. Sage Publications, Thousand Oaks
- Margolis B (2007) SOA for the business developer: concepts, BPEL, and SCA. Mc Press, Lewisville
- Marozzo F, Talia D, Trunfio P (2011) A cloud framework for parameter sweeping data mining applications. In: Proceedings of the 3rd IEEE international conference on cloud computing technology and science (CloudCom'11). IEEE Computer Society Press, Athens, pp 367–374. ISBN:978-0-7695-4622-3
- Marozzo F, Talia D, Trunfio P (2015) Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms. *Concurr Comput Pract Exp* 27(17):5214–5237
- Marozzo F, Talia D, Trunfio P (2016) A workflow management system for scalable data mining on clouds. *IEEE Trans Serv Comput* PP(99):1–1
- Talia D, Trunfio P, Marozzo F (2015) Data analysis in the cloud. Elsevier. ISBN:978-0-12-802881-0
- Valiant LG (1990) A bridging model for parallel computation. *Commun ACM* 33(8):103–111
- WFMC T (1999) Glossary, document number WFMC, issue 3.0. TC 1011
- Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS, Foster I (2011) Swift: a language for distributed parallel scripting. *Parallel Comput* 37(9):633–652
- Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, Soiland-Reyes S, Dunlop I, Nenadic A, Fisher P et al (2013) The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Res* 41(W1):W557–W561
- Wozniak JM, Wilde M, Foster IT (2014) Language features for scalable distributed-memory dataflow computing. In: 2014 fourth workshop on data-flow execution models for extreme scale computing (DFM). IEEE, pp 50–53

Worklet

► Component Benchmark

Y

Yardstick

- ▶ Metrics for Big Data Benchmarks

YCSB

Steffen Friedrich and Norbert Ritter
Department of Informatics, University of
Hamburg, Hamburg, Germany

Definitions

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source framework originally developed by Yahoo! to characterize and compare the performance of different *NoSQL database systems* (Cooper et al. 2010). YCSB can be classified as a *CRUD benchmark*, because the supported workloads are limited to a mix of CRUD (create, read, update, and delete) and scan operations.

It is a highly generic general-purpose benchmark that can easily be extended to support additional databases as it only requires the implementation of this simple CRUD interface.

The database system under test is stressed with the customizable mix of operations by a configurable number of client threads. At the same time, request latency and throughput (operations per seconds) are measured. The actual database

records consist of a number of synthetically generated string attributes.

Both *workload generation* and *data generation* are highly configurable. The workload generator randomly selects the CRUD operations according to their respective probabilities specified by the user. Different random distributions (uniform, exponential, Zipfian, or latest) can be selected. These determine the selection of the keys to work on, the number of records for the scan operation, and the length of the generated string attributes. YCSB comes with a set of basic workloads, which are based on typical web application workloads observed at Yahoo!

Furthermore, it is possible to configure runtime parameters like the number of operations to perform, the maximum execution time, the number of records to insert, and the desired target throughput.

Its extensibility and the generic workloads make YCSB suitable to explore general tradeoffs in a wide range of storage systems and finally led to its status as the de facto standard for comparative performance evaluation of NoSQL database systems.

Overview

Since the 1980s, relational database management systems (RDBMSs) have been regarded as the “one-size-fits-all” solution for persisting and querying data, and they have reached an

unmatched level of reliability, stability, and standardization through decades of development. Especially the standard query language (SQL) made it easy to develop domain-specific benchmarks with real application workloads, such as the famous TPC-C (2010). The TPC-C belongs to a series of benchmark specifications of the Transaction Processing Performance Council. Most of these specifications have in common that they expect the database system under test to support ACID-compliant transactions.

To achieve horizontal scalability and high availability, NoSQL database systems usually forgo the relational data model, join operations, transactions, and the consistency guarantees offered by traditional RDBMSs. Therefore, the TPC standards for benchmarking relational database systems can no longer be applied to the NoSQL systems. In consequence, YCSB was developed in order to enable a performance comparison between them.

However, some recent TPC specifications also take NoSQL database systems into account. For example, the implementation of the *TPCx-IoT* is even based on YCSB.

Scientific Fundamentals

The Benchmark Handbook for Database and Transaction Systems edited by Jim Gray (1993) can be regarded as the first reference work of database benchmarking. It includes several early benchmarking efforts for RDBMS, such as the first TPC standards and the Wisconsin benchmark. While the former became established as the standard for relational database system benchmarking, the Wisconsin benchmark had to accept a lot of criticism, partly because it did not model a real application workload. Nevertheless, it has been extensively used in research on parallel relational database systems for shared-nothing architectures. In this context, DeWitt (1993) already described the three fundamental metrics size-up, scale-up, and speedup, which cover important nonfunctional properties of distributed database systems. YCSB adapts them to the concept of different benchmark tiers:

- **Tier 1 – Performance** measures the latency of the system as the throughput is increased in different runs. This experiment class is very similar to the **sizeup** metric of the Wisconsin benchmark.

- **Tier 2 – Scalability and Elasticity**

Scalability measures the latency while the capacity is increased between different workload runs. Usually, in each step, another server is added and more data is loaded into the system. This corresponds to the **scaleup** metric described by DeWitt. **Elasticity** measures the impact on latency and throughput when one or more servers are added to the database system while the workload is running. This is similar to the **speedup** metric.

Kuhlenkamp et al. (2014) provide an overview of scalability and elasticity experiments carried out in various research projects with YCSB. They classify them into finer categories given their specific experimental designs and repeat some benchmarks in order to verify results.

In addition to these two tiers, Cooper et al. (2010) also proposed **Tier 3 Availability** and **Tier 4 Replication** for future research. The availability tier measures the performance impact of failures on the system. So far, there have only been initial research efforts that consider single-server outages in their benchmarks (Fior et al. 2013; Rosselli et al. 2016). The replication tier, on the other hand, considers the performance impact of different replication techniques or configurations (Klems et al. 2012; Haughian et al. 2016).

A lot of research was done in the area of consistency measurement of replicated database systems. For this purpose, YCSB has been extended and has also been used as workload generator while conducting consistency measurements with specialized tools. Friedrich et al. (2014) give an overview of the various research projects.

Finally, the book *Cloud Service Benchmarking* by Bermbach et al. (2017) can be used as a modern reference book in which the fundamentals of database benchmarking are adapted to today's cloud and Big Data requirements.

Beyond CRUD Operations

YCSB is criticized for not considering functionality beyond that of simple key-value stores and for abstracting from many differences, such as data models. In response to these criticisms, many extensions have been developed.

For example, YCSB + T wraps the CRUD operations into transactional context to model a simple “closed economy workload” (Dey et al. 2014). It was used to evaluate the performance of a generic multi-item transaction protocol for key-value stores. Pilman et al. (2017), on the other hand, added some additional scan queries (e.g., with aggregation) to benchmark their proposed SQL-over-NoSQL architecture. In this context, they examine the performance of various NoSQL systems offering a simple scan operator.

Other extensions allow benchmarking advanced query capabilities of certain classes of NoSQL databases. These include XGDBBench for graph databases (Dayarathna and Suzumura 2012) and YCSB-TS to measure the performance of time-series database functionalities.

Improving YCSB

Given the fundamental requirements of database system benchmarks, namely, relevance, simplicity, portability, and scalability (Gray 1993), YCSB favors portability and simplicity over relevance. Unlike the TPC benchmarks, it does not model a real application workload and is therefore not very relevant for one particular application domain. But its generic workloads and the simple CRUD interface ensure the highest portability. However, there are some research efforts to improve the relevance and scalability aspect.

Patil et al. (2011) have added a coordination service in their extension called YCSB++, which distributes the originally centralized benchmarking framework across several benchmark machines. The extension KV-replay (Boza et al. 2017) includes the option to replay realistic workloads by reading in trace files of real application workloads. Friedrich et al. (2017), on the other hand, implemented a load generator based on an *open system model*, in addition to YCSB’s implementation following a *closed system model*.

In the closed system model, a fixed number of threads repeatedly request the system, whereby a new request can only be sent after a response. In the open system model, requests are independent of each other and can be issued without any constraints linked to responses. Since the implementation of an open system model is similar to modern asynchronous web frameworks, this leads to more realistic web application workloads.

Key Applications

Currently, YCSB is used for experimental evaluation in almost all scientific work in the field of distributed data management. It is also extensively used in studies that investigate the performance behavior of certain system or protocol designs. For example, Harding et al. (2017) recently compared the performance characteristics of six classic and modern concurrency control protocols for distributed transactions. Other research activities deal with general quality tradeoffs like the consistency/latency tradeoff of NoSQL database systems (Rahman et al. 2017).

Of course, YCSB is also used by NoSQL vendors to carry out comparative studies for marketing and advertising purposes. The presented results should always be viewed critically, as the experimental design and the presentation of the results are usually chosen in favor of the advertised system.

URL to Code

- YCSB:
<https://github.com/brianfrankcooper/YCSB>
- YCSB+T:
<https://github.com/akon-dey/YCSB>
- YCSB-TS:
<https://github.com/TSDBBench/YCSB-TS>
- XGDBBench:
<https://github.com/miyurud/XGDBench>
- KV-replay:
<https://github.com/ebozag/KV-replay>

Cross-References

- ▶ [CRUD Benchmarks](#)
- ▶ [NoSQL Database Systems](#)
- ▶ [System Under Test](#)

References

- Bermbach D, Wittern E, Tai S (2017) Cloud service benchmarking: measuring quality of cloud services from a client perspective. Springer, Cham
- Boza EF, San-Lucas C, Abad CL, Viteri JA (2017) Benchmarking key-value stores via trace replay. In: 2017 IEEE international conference on cloud engineering (IC2E), pp 183–189
- Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on cloud computing, SoCC’10. ACM, New York, pp 143–154
- Dayarathna M, Suzumura T (2012) Xgdbench: a benchmarking platform for graph stores in exascale clouds. In: 4th IEEE international conference on cloud computing technology and science proceedings, pp 363–370
- DeWitt DJ (1993) The Wisconsin benchmark: past, present, and future. In: Gray J (ed) The benchmark handbook. Morgan Kaufmann, San Mateo
- Dey A, Fekete A, Nambiar R, Röhm U (2014) YCSB + T: benchmarking web-scale transactional databases. In: Proceedings of international workshop on cloud data management (CloudDB’14), Chicago
- Fior AG, Meira JA, de Almeida EC, Coelho RG, Fabro MDD, Traon YL (2013) Under pressure benchmark for DDBMS availability. JIDM 4(3):266–278
- Friedrich S, Wingerath W, Gessert F, Ritter N (2014) NoSQL OLTP benchmarking: a survey. In: 44. Jahrestagung der Gesellschaft für Informatik, Informatik 2014, Big data – Komplexität meistern, 22–26 Sept 2014, Stuttgart, pp 693–704
- Friedrich S, Wingerath W, Ritter N (2017) Coordinated omission in NoSQL database benchmarking. In: Datenbanksysteme für Business, Technologie und Web (BTW), Stuttgart. Workshopband, GI Bonn, pp 215–225
- Gray J (ed) (1993) The benchmark handbook for database and transaction systems, 2nd edn. Morgan Kaufmann, San Mateo
- Harding R, Van Aken D, Pavlo A, Stonebraker M (2017) An evaluation of distributed concurrency control. Proc VLDB Endow 10(5):553–564
- Haughian G, Osman R, Knottenbelt WJ (2016) Benchmarking replication in Cassandra and MongoDB NoSQL datastores. In: Database and expert systems applications – proceedings of the 27th international conference, DEXA 2016, part II, Porto, 5–8 Sept 2016, pp 152–166
- Klems M, Bermbach D, Weinert R (2012) A runtime quality measurement framework for cloud database service systems. In: Proceedings of the 2012 eighth international conference on the quality of information and communications technology, QUATIC’12. IEEE Computer Society, Washington, DC, pp 38–46
- Kuhlenkamp J, Klems M, Röss O (2014) Benchmarking scalability and elasticity of distributed database systems. Proc VLDB Endow 7(12):1219–1230
- Patil S, Polte M, Ren K, Tantisiriroj W, Xiao L, López J, Gibson G, Fuchs A, Rinaldi B (2011) YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: Proceedings of the 2nd ACM symposium on cloud computing, SOCC’11. ACM, New York, pp 9:1–9:14
- Pilman M, Bocksrocker K, Braun L, Marroquín R, Kossmann D (2017) Fast scans on key-value stores. Proc VLDB Endow 10(11):1526–1537
- Rahman MR, Tseng L, Nguyen S, Gupta I, Vaidya N (2017) Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores. ACM Trans Auton Adapt Syst 11(4):20:1–20:36
- Rosselli M, Niemann R, Ivanov T, Tolle K, Zicari RV (2016) Benchmarking the availability and fault tolerance of Cassandra. In: Rabl T, Nambiar R, Baru C, Bhandarkar M, Poess M, Pyne S (eds) Big data benchmarking. WBDB 2015. Lecture notes in computer science, vol 10044. Springer, Cham, pp 87–95
- TPC-C (2010) Benchmark specification. <http://www.tpc.org/tpcc>